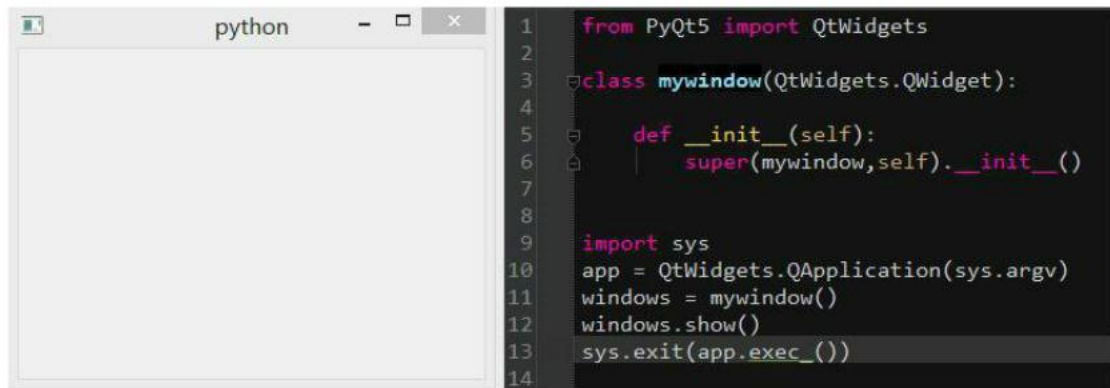


pyqt5&python Gui 入门教程 (1) 第一个窗口 (1)



第一个窗口和代码详细注释:

```
from PyQt5 import QtWidgets
```

#从 PyQt 库导入 QWidget 通用窗口类

```
class mywindow(QtWidgets.QWidget):
```

#自己建一个 mywindows 类, 以 class 开头, mywindows 是自己的类名,

(QtWidgets.QWidget) 是继承 QtWidgets.QWidget 类方法,

定义类或函数不要忘记 ':' 符号, 判断语句也必须以 ':' 结尾!

```
    def __init__(self):
```

#def 是定义函数 (类方法) 了, 同样第二个 __init__ 是函数名

(self) 是 pyqt 类方法必须要有的, 代表自己, 相当于 java, c++ 中的 this

#其实 __init__ 是析构函数, 也就是类被创建后就会预先加载的项目

```
        super(mywindow, self).__init__()
```

#这里我们要重载一下 mywindows 同时也包含了 QtWidgets.QWidget 的预加载项

```
import sys
```

```
app = QtWidgets.QApplication(sys.argv)

#pyqt 窗口必须在 QApplication 方法中使用，

#要不然会报错 QWidget: Must construct a QApplication before a QWidget

windows = mywindow()

# 生成过一个实例（对象），windows 是实例（对象）的名字，可以随便起！

# mywindows（）是我们上面自定义的类

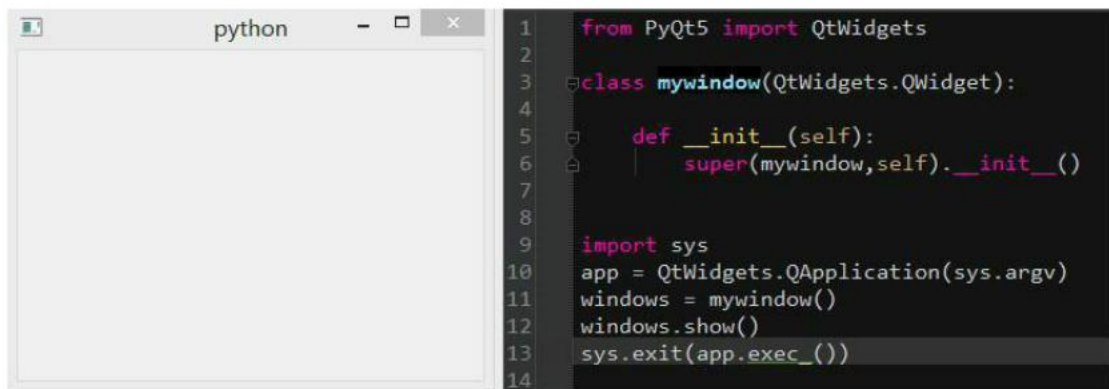
windows.show()

#有了实例，就得让他显示这里的 show() 是 QWidget 的方法，用来显示窗口的！

sys.exit(app.exec_())

#启动事件循环
```

pyqt5&python Gui 入门教程（2） 第一个窗口（2）



上图是第一篇教程，下面的显示效果都一样，我们来看看有什么不同

```

from PyQt5 import QtWidgets

class fristwindows(QtWidgets.QWidget):
    def __init__(self):
        super(fristwindows,self).__init__()

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    new = fristwindows()
    new.show()
    sys.exit(app.exec_())

```

- 1、类的名字、实例的名字都换了，
- 2、多了一个 if `__name__ == "__main__":`：以及下面的代码缩进了，层次改变了

```

from PyQt5 import QtWidgets

class fristwindows(QtWidgets.QWidget):
    def __init__(self):
        super(fristwindows,self).__init__()

def mainwindows():
    import sys
    app = QtWidgets.QApplication(sys.argv)
    new = fristwindows()
    new.show()
    sys.exit(app.exec_())

if __name__ == "__main__":
    mainwindows()

```

- 1、我们把结尾的 5 句代码，单独建立了一个函数
- 2、然后直接调用函数
- 3、注意两个 def 的缩进，第一个 def 缩进了代表是在 class 里面，第二个和 class 平齐，则是在外面。

可以看到显示效果是一样，我们却有很多办法去实现。

知识点:

1、if `__name__ == "__main__"`: 是代表如果这个文件是主程序这运行下面的代码, 如果是被别的程序文件调用的话, 则运行下面的代码。

2、`__init__` 方法在类的一个对象被建立时, 马上运行。这个方法可以用来对你的对象做一些你希望的 初始化 。注意, 这个名称的开始和结尾都是双下划线。

3、生成实例 (对象) 必须以类名 (), 别忘记了 ()

4、类中的函数 (方法) 必须有 `self`, 是代表属于这个实例 (对象) 本身持有的, 而外部定义的函数则不需要。

整个流程:

先导入 PyQt5 中的 QtWidgets 通用窗口库, 通过继承 QtWidgets.QWidget 来定义自己的窗口。然后生成一个对象 (实例化), 再调用 QWidget 的 `show()` 方法来显示这个窗口。

pyqt5&python Gui 入门教程 (3) 第一个窗口 (3)

frist.py 文件

```
from PyQt5 import QtWidgets
from frist import fristwindows

if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    new = fristwindows()
    new.show()
    sys.exit(app.exec_())
```

second.py 文件

这是两个文件，第一个文件和之前一样我们只录入的上半截，第二个文件我们保留了下半截，

也就是将一个文件分成两个文件，从 second.py 文件调用 frist.py 来显示窗口的目的。

注意 second.py 文件加入了一句 `from frist import fristwindows` 即导入 first.py 的 fristwindows 类

是不是很熟悉，将自己的文件作为库导入进来，然后调用即可！

虽然都是显示窗口，我们这里已经学习了很多办法来显示它。代码区别也是很大，关键是灵活运用。

知识点：

1、库，这个例子是将我们自己的文件作为库导入，也可以称为自定义库。

注意不用.py 结尾，pyqt 会自动识别。

2、frist.py 的 firstwindows 的代码就是显示一个窗口用的，但是没有将它实例化，并在 QApplication 中运行。

所以运行 frist.py 是不会显示任何窗口的。

我们将显示 firstwindows 的方法写在了第二个文件中，达到逻辑与界面分离的效果。

3、QApplication 相当于 main 函数，也就是整个程序（有很多文件）的主入口函数。

对于一个 Gui 程序必须至少有一个这样的实例来让程序运行。

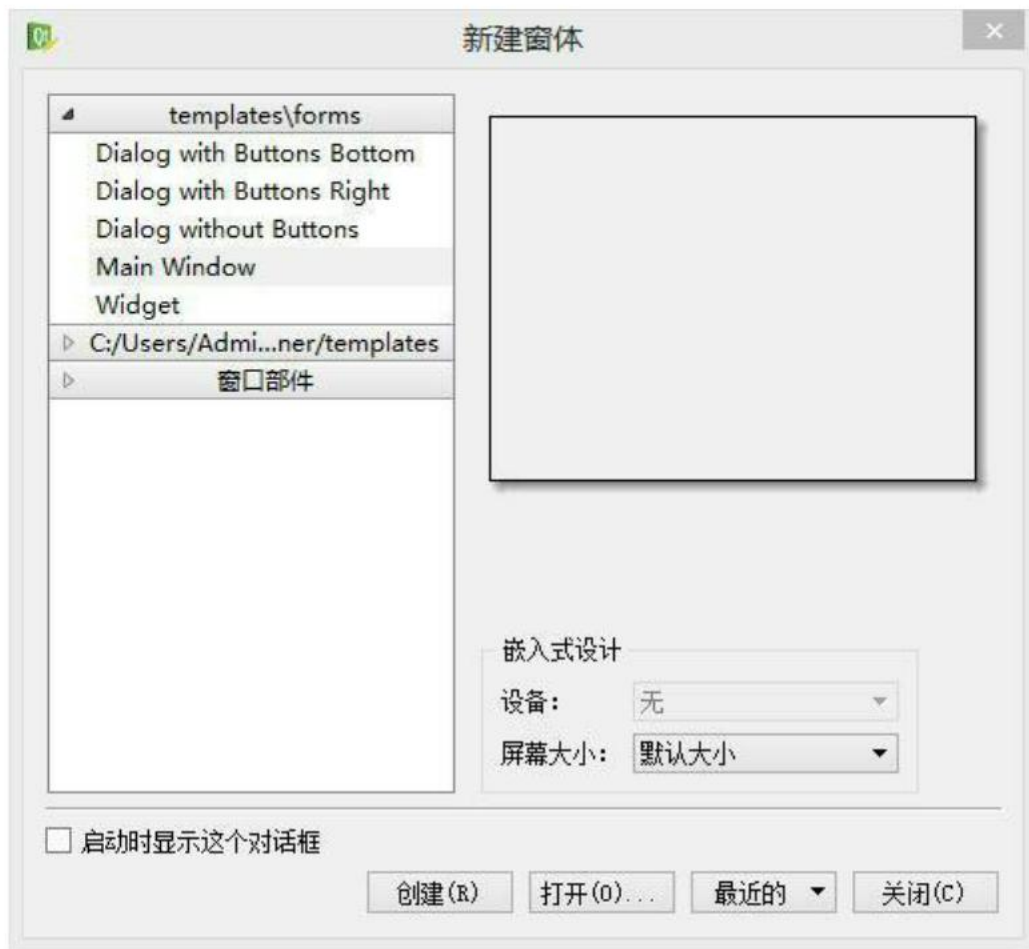
4、最后一句是调用 sys 库的 exit 退出方法，退出条件（参数）是 `app.exec_()` 也就是整个窗口关闭。

其实入门没什么难度，只是对一些基本概念的掌握和了解，本人也是菜鸟，写的不是很好，只是希望能和更多有兴趣爱好新手一起交流学习。

PyQt5&python Gui 入门教程（4） 初探 Qt Designer 设计师

网上很多教程都是代码，对于新手看起来就很头疼，pyqt 同样为我们提供了 Qt designer 来设计窗口界面，

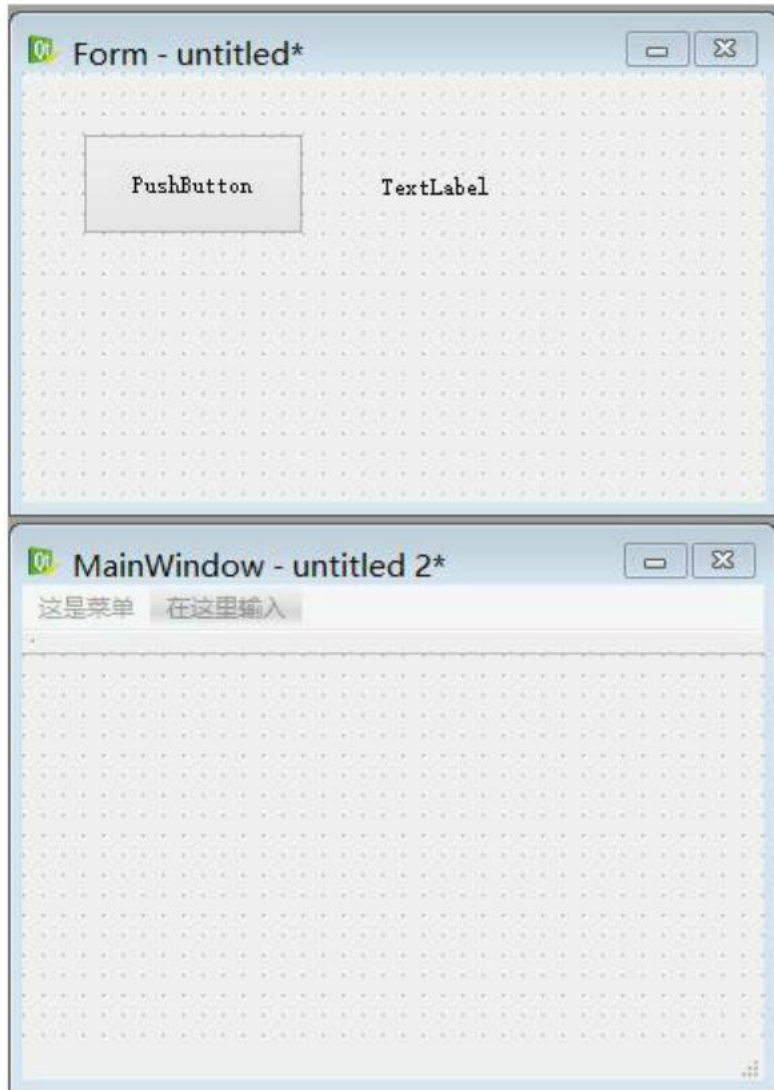
用起来也非常方便，对于新手我们应该善用它，虽然做出来的界面不那么华丽，但至少可以做出个像样的窗口来。



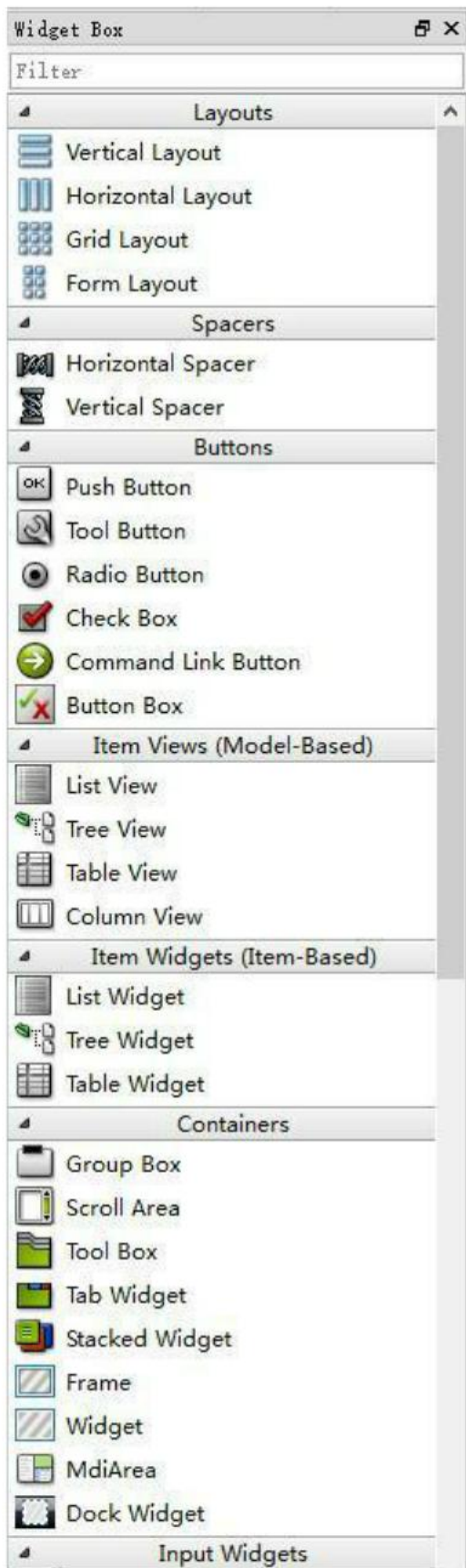
打开 PyQt5 的 Qt Designer，会自动弹出新建窗体对话框，

对于我们最常用的就是 Widget 通用窗口类，还有个 MainWindows 顾名思义主窗口。

PyQt5 的 Widget 被分离出来，似乎用来替代 Dialog，并将 Widget 放入了 QtWidget 模块（库）中，PyQt4 是 QtGUI。



这是一个Widget 和 MainWindows，从界面上看起来没有什么，只是 MainWindows 默认添加了菜单栏、工具栏和状态栏等。



默认左边是控件栏，提供了很多空间类，我们可以直接拖放到 widget 中看到效果，点窗体—预览 (Ctrl+R)

每个空间都有自己的名称，提供不同的功能，比如常用的按钮、输入框、单选、文本框等等，

对象查看器

对象

类

MainWindow

QMainWindow

centralwidget

QWidget

pushButton

QPushButton

menubar

QMenuBar

属性编辑器

Filter

+

-

MainWindow : QMainWindow

属性

值

QObject

objectName

MainWindow

QWidget

enabled

☒

geometry

[(0, 0), 414 x 346]

X

0

Y

0

宽度

414

高度

346

sizePolicy

[Preferred, Preferre...

水平策略

Preferred

资源浏览器

Filter

<resource root>

信号/槽编辑器

+

-

发送者

信号

接收者

槽

<

>

动作编辑器

Filter

名称

使用

文本

快捷键

<

>

右边是对窗口及控件的各种调整、设置、添加资源（列如:图片）、动作。

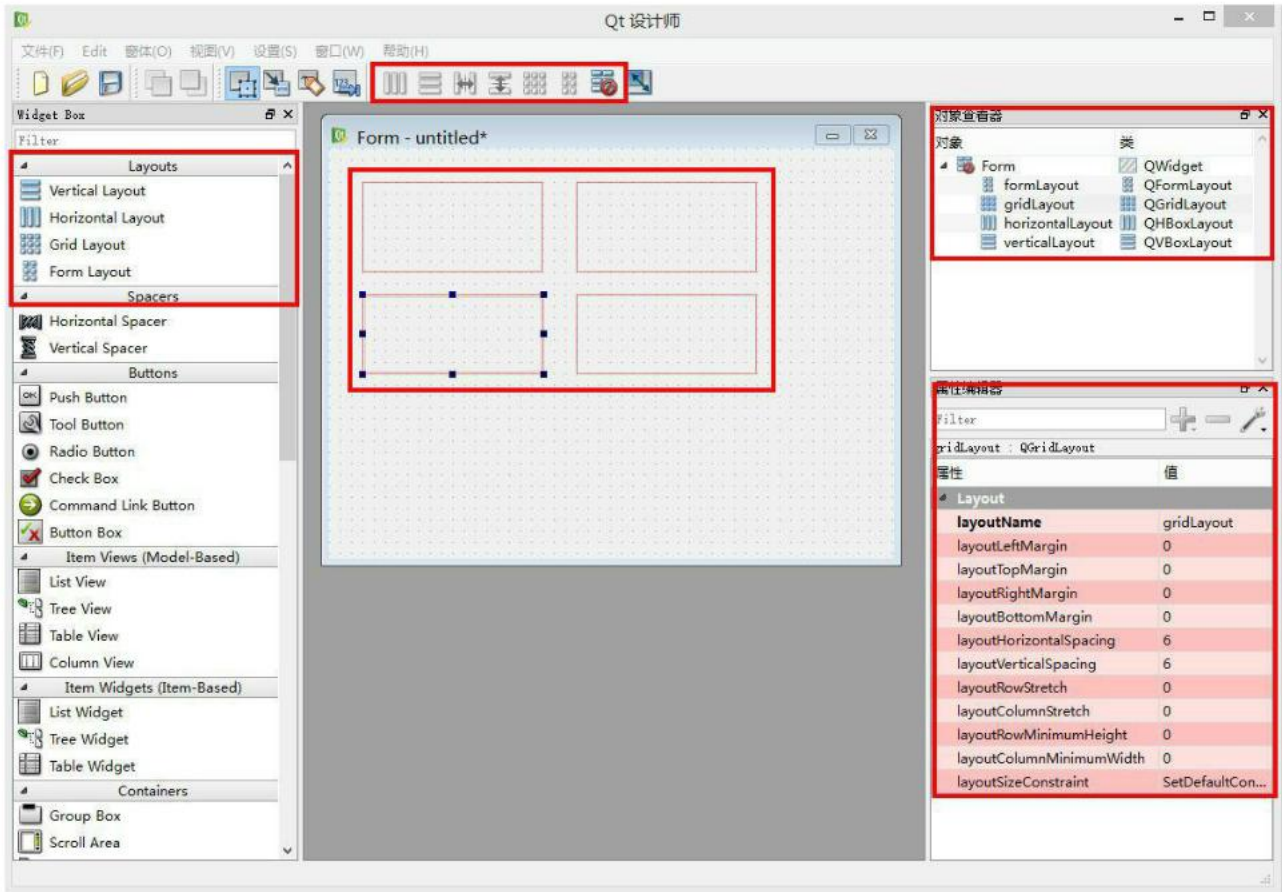
还可以直接编辑 Qt 引以为豪的信号槽（signal 和 slot）。

有了 Qt Designer 使得我们在程序设计中更快的能开发设计出程序界面，避免了用纯代码来写一个窗口的繁琐，

同时 PyQt 支持界面与逻辑分离，这对于新手来说无疑是个最大的福音，当然要做出的华丽的界面还是要学代码的。

至少 Qt Designer 为我们提供了一些解决方法，另外我们也可以通过 Qt Designer 生成的代码来学习一些窗口控件的用法。

PyQt5&python Gui 入门教程（5）Qt Designer 窗口布局 Layouts（1）



Qt Designer 窗口布局 Layouts 提供了四种布局方法，他们是：

Vertical Layout 纵向布局

Horizontal Layout 横向布局

Grid Layout 栅格布局

Form Layout 在窗体布局中布局

前三种是我们经常会用到的，我们将布局 Layouts 拖动到窗体上会有红色框来显示，

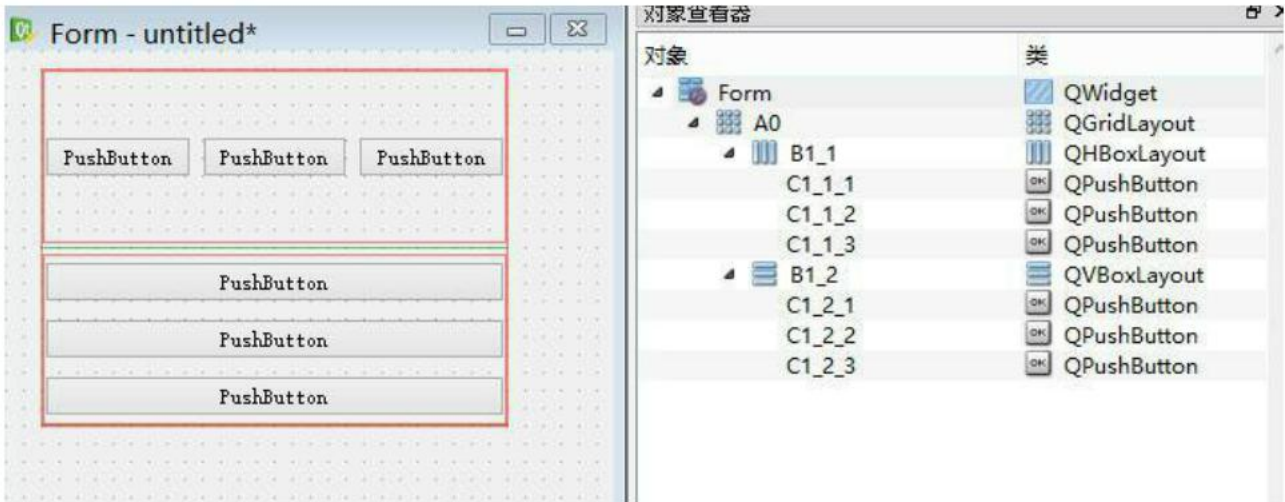
Layout 的一些属性可以通过属性编辑器来控制，一般包括：

上下左右边距间隔，空间之间间隔等。

在我们使用布局之前，我们得对层次要有个了解，在程序设计中一般用父子关系来表示。

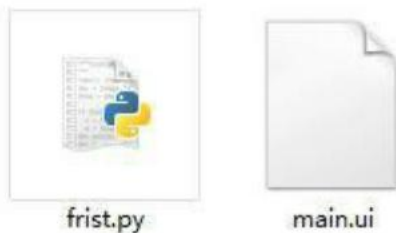
当然有过平面设计经验的童鞋对分层应该有所了解，这里我们还需要将层分成层次。

其实就像 python 中规定的代码缩进量代表不同层次的道理差不多。



从对象查看器中我们可以方便的看出窗体（Form）--布局（Layout）--控件（这里是 QPushButton 按钮）之间的层次关系。

Form 窗口一般作为顶层显示，然后使用 Layout 将控件按照我们想要的方式规划开来。



这里要注意一下，Qt Designer 设计出来的文件默认为 ui 文件，里面包含的类 css 布局设计语言，

如果想要查看代码我们还需要将它转换（编译）成 py 文件，我们可以使用一条 DOS 命令来完成

```
D:\Python33\Lib\site-packages\PyQt5\pyuic5.bat mian.ui -o frist.py
```

更实用的转换命令可以将当前文件夹下所有 ui 转换成 py 文件：

```
for /f "delims=" %%i in ('dir /b /a-d /s *.ui') do
D:\Python33\Lib\site-packages\PyQt5\pyuic5.bat %%i -o %%i.py
```

PyQt 支持用 LoadUi 方法直接加载 ui 文件，当然我们通过转换后可以方便学习 PyQt 窗体控件的源代码。

下一篇我们分析一下 Qt Designer 布局的源代码

PyQt5&python Gui 入门教程（6）Qt Designer 窗口布局 Layouts（2）

```
from PyQt5 import QtCore, QtGui, QtWidgets    #导入模块

class Ui_Form(object):    #创建窗体类 继承自object
    def setupUi(self, Form):    #创建setUi函数
        Form.setObjectName("Form")    #设置窗体名 注: ObjectName
        Form.resize(340, 365)    #设置窗体大小
        self.retranslateUi(Form)#加载retranslateUi函数
        QtCore.QMetaObject.connectSlotsByName(Form)    #关联信号槽

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate    #国际化支持
        Form.setWindowTitle(_translate("Form", "Form"))    #设置窗口标题
```

首先我们来看一下，我们创建一个空白 Widget 窗体，Qt Designer 都为我们做了些什么？

嗯，比我们第一个窗体，多了那么几行代码，默认转换后的 py 文件还不能直接显示出效果，下面我们慢慢学习。


```

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(389, 279)
        self.gridLayoutWidget = QtWidgets.QWidget(Form) #创建一个栅格布局，并放入到窗体Form
        self.gridLayoutWidget.setGeometry(QtCore.QRect(50, 50, 281, 181)) #设置栅格布局大
        self.gridLayoutWidget.setObjectName("gridLayoutWidget") #设置栅格布局ObjectName
        self.gridLayout = QtWidgets.QGridLayout(self.gridLayoutWidget) # #创建一个栅格布局
        self.gridLayout.setContentsMargins(0, 0, 0, 0) #设置栅格布局边距
        self.gridLayout.setObjectName("gridLayout") #设置栅格布局ObjectName

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))

```

这里我们只加入一个栅格布局，然后.....Qt Designer 居然给我们多了 6 行代码

其实 Qt Designer 生成的代码（编译后）还是非常给力的，至少比我们自己写的规范多了。

这些代码或许不需要我们自己写，但至少我们能看得明白。

其中最主要的两句代码是：

```
self.gridLayoutWidget = QtWidgets.QWidget(Form)
```

```
self.gridLayout = QtWidgets.QGridLayout(self.gridLayoutWidget)
```

从 Qt Designer 中可以看到栅格布局默认创建了两个对象，QtWidgets.QWidget 和 QtWidgets.QGridLayout。

注意(Form)和(self.gridLayoutWidget)参数，这里可以理解为放置到相应的对象里面。

```
from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(336, 275)
        self.gridLayoutWidget = QtWidgets.QWidget(Form)
        self.gridLayoutWidget.setGeometry(QtCore.QRect(50, 50, 241, 181))
        self.gridLayoutWidget.setObjectName("gridLayoutWidget")
        self.gridLayout = QtWidgets.QGridLayout(self.gridLayoutWidget)
        self.gridLayout.setContentsMargins(0, 0, 0, 0)
        self.gridLayout.setObjectName("gridLayout")
        self.verticalLayout = QtWidgets.QVBoxLayout() #创建（生成）一个横向布局
        self.verticalLayout.setObjectName("verticalLayout")
        self.pushButton = QtWidgets.QPushButton(self.gridLayoutWidget) #创建一个按钮1, 并放置到self.gridLayoutWidget
        self.pushButton.setObjectName("pushButton")
        self.verticalLayout.addWidget(self.pushButton) #把按钮1加入横向布局
        self.pushButton_2 = QtWidgets.QPushButton(self.gridLayoutWidget) #再创建一个按钮2, 并放置到self.gridLayoutWidget
        self.pushButton_2.setObjectName("pushButton_2")
        self.verticalLayout.addWidget(self.pushButton_2) #把按钮2加入横向布局
        self.gridLayout.addLayout(self.verticalLayout, 0, 0, 1, 1) #把横向布局加入到栅格布局

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "Form"))
        self.pushButton.setText(_translate("Form", "PushButton")) #设置按钮文字
        self.pushButton_2.setText(_translate("Form", "PushButton")) #设置按钮文字
```

Qt Designer 果然做的比我们想想的多，下面我们再通过简化后的代码来对比一下，可能跟容易理解一些。

```
from PyQt5 import QtWidgets

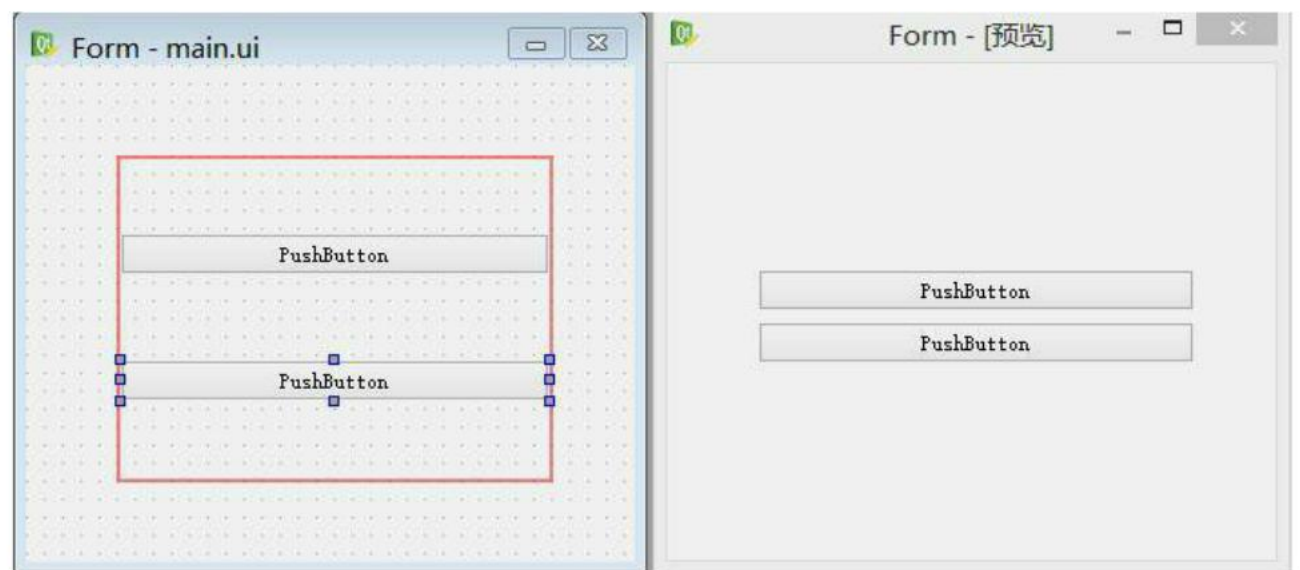
class FristWindows(QtWidgets.QWidget):
    def __init__(self):
        super(FristWindows, self).__init__()
        self.resize(336, 275) #设置窗体大小
        self.gridLayout = QtWidgets.QGridLayout() #创建一个栅格布局对象
        self.verticalLayout = QtWidgets.QVBoxLayout() #创建一个横向布局对象
        self.pushButton_1 = QtWidgets.QPushButton() #创建一个按钮
        self.pushButton_2 = QtWidgets.QPushButton() #创建一个按钮
        self.verticalLayout.addWidget(self.pushButton_1) #将按钮1放入横向布局
        self.verticalLayout.addWidget(self.pushButton_2) #将按钮2放入横向布局
        self.gridLayout.addLayout(self.verticalLayout, 0, 0, 1, 1) #将横向布局放入栅格布局
        self.setLayout(self.gridLayout) #设置窗体默认显示布局
        self.pushButton_1.setText("按钮1") #设置按钮文字
        self.pushButton_2.setText("按钮2") #设置按钮文字
```

注意:Qt Designer 首先会见栅格布局作为一个 widget 放入窗体, 并且不影响窗体大小。

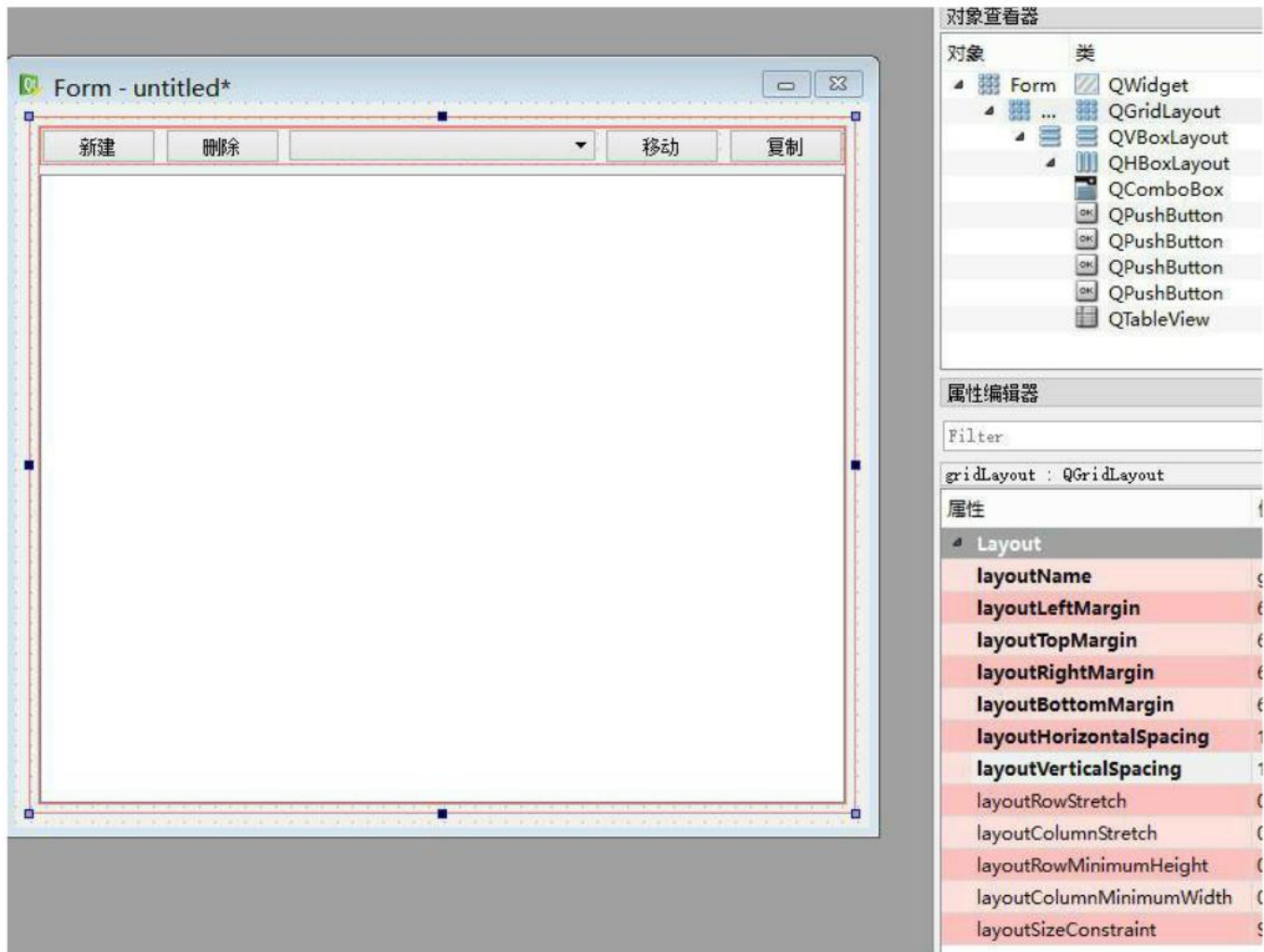
栅格布局一般作为顶层布局使用, 所以简化后的代码有个 `self.setLayout`
(`self.gridLayout`)

同时 Qt Designer 会加入一些基本设置, 默认信号槽声明, 国际化语言支持等等, 确实比我们自己写的要好的多。

当然我们需要了解其中的奥秘。



左边是 Qt Designer 设计效果, 右边是显示效果。



再来个稍微复杂点的

知识点：

记得在 Qt Designer 中窗体的 layout 层次可以通过对象查看器来查看，layout 的一些设置可以通过属性编辑器来修改。

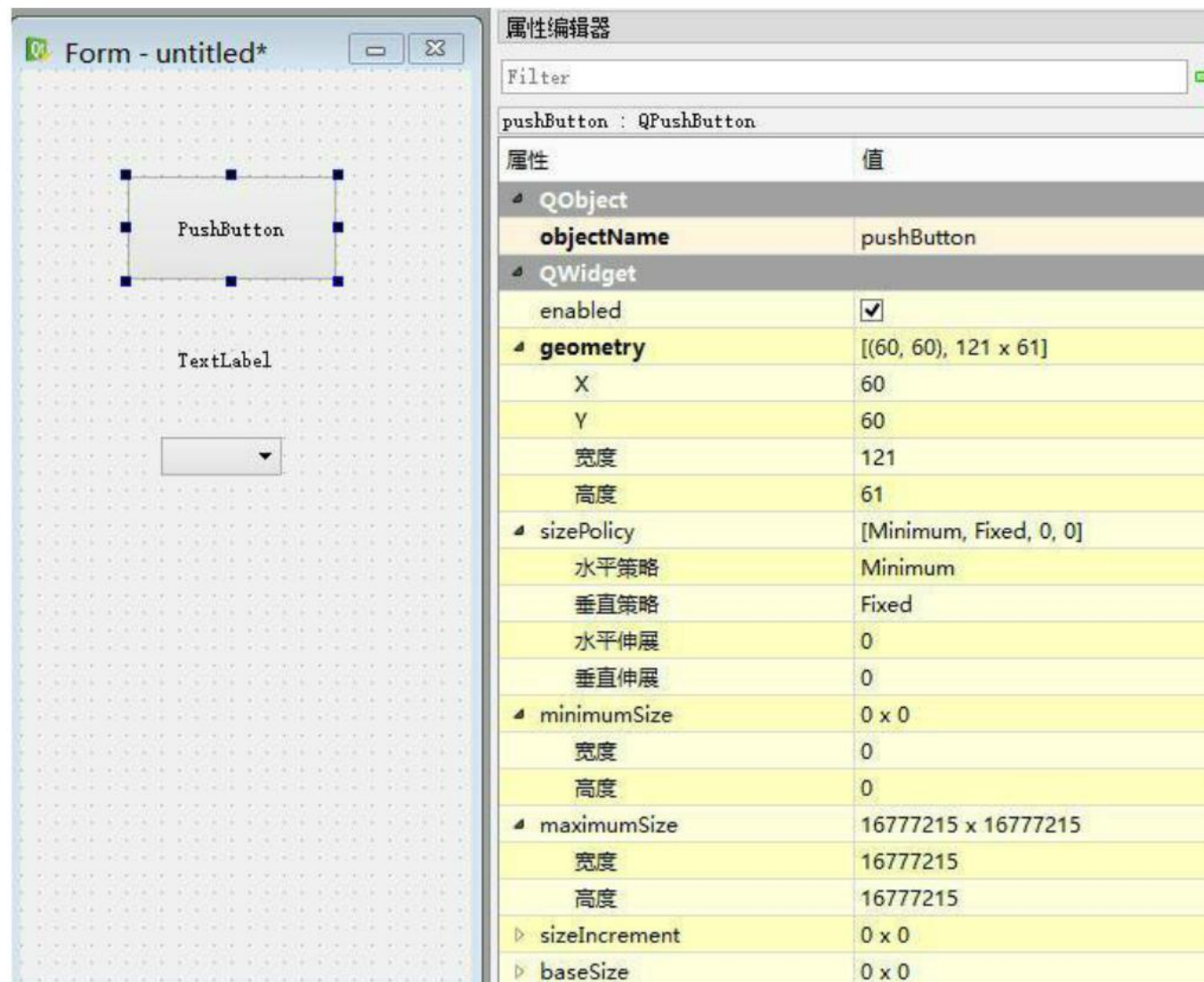
通常我们使用栅格布局作为顶层布局，将控件放置好之后可以通过右键—布局—栅格布局，将布局充满整个窗体。

我们可以先放入控件，然后 ctrl 选中多个控件，然后点击工具栏上快速布局工具进行布局。

在 QMainWindow 中默认会有个 centralWidget 布局也是继承自 QWidget，表示窗口的中央部分。

在接下来的教程你会看到 QMainWindow 的使用技巧。

PyQt5&python Gui 入门教程（7） Qt Designer 控件的一些通用属性



在 Qt Designer 中的右边为我们提供了窗体、控件、布局的属性编辑功能。

比较常用的有：

objectName 控件对象名称 例：`anniu = QtWidgets.QPushButton()` 等号前面的那个名字

geometry 相对坐标系

sizePolicy 控件大小策略

minimumSize 最小宽度、高度

maximumSize 最大宽度、高度 如果想让窗体或控件固定大小，可以将 mini 和 max 这两个属性设置成一样的数值

font 字体

cursor 光标

windowTitle 窗体标题

windowsIcon / icon 窗体图标/控件图标

iconSize 图标大小

toolTip 提示信息

statusTip 任务栏提示信息

text 控件文字

shortcut 快捷键




```

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(200, 200) #窗体大小
        Form.setMinimumSize(QtCore.QSize(200, 200)) #窗体最小大小
        Form.setMaximumSize(QtCore.QSize(200, 200)) #窗体最大大小 与上面一样，即不可缩放
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("../ico/1 (1).png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
        Form.setWindowIcon(icon) #设置窗体图标
        self.pushButton = QtWidgets.QPushButton(Form) #创建按钮
        self.pushButton.setGeometry(QtCore.QRect(50, 60, 100, 50)) #按钮坐标
        self.pushButton.setMinimumSize(QtCore.QSize(100, 50)) #按钮最小大小
        self.pushButton.setMaximumSize(QtCore.QSize(100, 50)) #按钮最大大小
        font = QtGui.QFont() #设置字体
        font.setPointSize(12)
        font.setBold(True)
        font.setItalic(False)
        font.setUnderline(False)
        font.setWeight(75)
        self.pushButton.setFont(font)
        self.pushButton.setCursor(QtGui.QCursor(QtCore.Qt.BusyCursor)) #设置光标
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap("../ico/ironman.png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.pushButton.setIcon(icon1) #设置按钮图标
        self.pushButton.setIconSize(QtCore.QSize(42, 42))
        self.pushButton.setObjectName("pushButton")

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "这是窗体")) #设置窗体标题
        self.pushButton.setText(_translate("Form", "按钮")) #设置按钮文字
        self.pushButton.setShortcut(_translate("Form", "Ctrl+A")) #设置按钮快捷键

```

PyQt5&python Gui 入门教程 (8) Qt Designer 界面
与逻辑分离

```

from PyQt5 import QtCore, QtGui, QtWidgets

class Ui_Form(object):
    def setupUi(self, Form):
        Form.setObjectName("Form")
        Form.resize(200, 200) #窗体大小
        Form.setMinimumSize(QtCore.QSize(200, 200)) #窗体最小大小
        Form.setMaximumSize(QtCore.QSize(200, 200)) #窗体最大大小 与上面一样，即不可缩放
        icon = QtGui.QIcon()
        icon.addPixmap(QtGui.QPixmap("../ico/1 (1).png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
        Form.setWindowIcon(icon) #设置窗体图标
        self.pushButton = QtWidgets.QPushButton(Form) #创建按钮
        self.pushButton.setGeometry(QtCore.QRect(50, 60, 100, 50)) #按钮坐标
        self.pushButton.setMinimumSize(QtCore.QSize(100, 50)) #按钮最小大小
        self.pushButton.setMaximumSize(QtCore.QSize(100, 50)) #按钮最大大小
        font = QtGui.QFont() #设置字体
        font.setPointSize(12)
        font.setBold(True)
        font.setItalic(False)
        font.setUnderline(False)
        font.setWeight(75)
        self.pushButton.setFont(font)
        self.pushButton.setCursor(QtGui.QCursor(QtCore.Qt.BusyCursor)) #设置光标
        icon1 = QtGui.QIcon()
        icon1.addPixmap(QtGui.QPixmap("../ico/ironman.png"), QtGui.QIcon.Normal, QtGui.QIcon.Off)
        self.pushButton.setIcon(icon1) #设置按钮图标
        self.pushButton.setIconSize(QtCore.QSize(42, 42))
        self.pushButton.setObjectName("pushButton")

        self.retranslateUi(Form)
        QtCore.QMetaObject.connectSlotsByName(Form)

    def retranslateUi(self, Form):
        _translate = QtCore.QCoreApplication.translate
        Form.setWindowTitle(_translate("Form", "这是窗体")) #设置窗体标题
        self.pushButton.setText(_translate("Form", "按钮")) #设置按钮文字
        self.pushButton.setShortcut(_translate("Form", "Ctrl+A")) #设置按钮快捷键

```

这是上一篇文章中用 Qt Designer 设计编译后生成的代码，但是想要让它显示出来，还是得费点功夫，

对于新手来说经常会碰壁，我也是折腾了好长时间才明白，原来还要做一下一些事情。



这是显示效果。

```
if __name__ == "__main__":  
    import sys  
    app = QtWidgets.QApplication(sys.argv)  
    widget = QtWidgets.QWidget()  
    ui = Ui_Form()  
    ui.setupUi(widget)  
    widget.show()  
    sys.exit(app.exec_())
```

如果我们在单个文件当中显示，还需要在结尾加入以上代码。

因为 Qt Designer 默认继承的 object 类，不提供 show() 显示方法，

所以我们生成一个 QWidget 对象来重载我们设计的 Ui_Form 类，达到显示效果。

在实际工作中，我们希望界面和代码分离，而防止破坏 Qt Designer 编译生成的代码，

我们可以这样做：


```

1  from PyQt5 import QtWidgets
2  from myDesigner import Ui_Form
3  import sys
4
5  if __name__ == "__main__":
6      app = QtWidgets.QApplication(sys.argv)
7      widget = QtWidgets.QWidget()
8      new = Ui_Form()
9      new.setupUi(widget)
10     widget.show()
11     sys.exit(app.exec_())
12

```

新建一个文件，导入我们设计的 myDesigner.py 文件，然后下面的代码相同即可。

但是这是在 main 方法中调用窗口，也不符合我们的实际工作中的要求。

我们还可以这样搞：

```

from PyQt5 import QtWidgets
from myDesigner import Ui_Form
import sys

class mydesignershow(QtWidgets.QWidget):#继承QWidget
    def __init__(self):
        super(mydesignershow,self).__init__()
        self.new = Ui_Form()          #创建实例
        self.new.setupUi(self)         #加载窗体

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myshow = mydesignershow()          #创建实例
    myshow.show()                     #使用QWidget的show()方法
    sys.exit(app.exec_())

```

默认的 Qt Designer 生成的代码是继承 object 类，所以默认编译生成的文件不能显示。

通过不同的方法，可以知道我们需要继承或重载 QWidget 这个类，来使用其中的 show 方法使窗口显示。

这也就是传说中的界面与代码分离，其实也没有想象中的那么难。

```

1  from PyQt5 import QtWidgets
2  from myDesigner import Ui_Form
3  import sys
4
5  class mydesignershow(QtWidgets.QWidget,Ui_Form):#继承QWidget和Ui_Form
6      def __init__(self):
7          super(mydesignershow,self).__init__()
8          self.setupUi(self)          #加载窗体
9
10     if __name__ == "__main__":
11         app = QtWidgets.QApplication(sys.argv)
12         myshow = mydesignershow()      #创建实例
13         myshow.show()                #使用QWidget的show()方法
14         sys.exit(app.exec_())
15

```

我们还可以通过同时继承 QWidget 和我们设计的 Ui_Form 类来简化代码，这就是传说中的多态？

之前我们见过，__init__是析构函数，也就是类被实例化（生成对象）之后默认加载的内容。

super().__init__() 超级加载，也就是我们需要同时加载 QWidget 中的内容供我们使用。

self.setupUi(self) self 是指自己，setupUi 也就是 Ui_Form 中的方法。

还记得之前说的：实例.方法(参数) self 在实例是代表自己这个类

myshow 是实例，mydesignershow() 是类，用=代表创建一个实例对象。

myshow.show() 然后调用对象中的方法，相当于调用类里面的函数。

PyQt5&python Gui 入门教程（9）Qt Designer 初探 信号槽

信号与槽作为 QT 的核心机制，往往给新手带来很多困扰，这也不是一天两天能搞懂学透的东西，

我们的宗旨是要会用，就算是简单的运用吧，学以致用才是正道，也是沧桑啊！

先入一些信号槽的基本介绍：

信号和槽是一种高级接口，应用于对象之间的通信，它是 QT 的核心特性，也是 QT 区别于其它工具包的重要地方。它为高层次的事件处理自动生成所需要的附加代码。在我们所熟知的很多 GUI 工具包中，窗口小部件 (widget) 都有一个回调函数用于响应它们能触发的每个动作，这个回调函数通常是一个指向某个函数的指针。但是，在 QT 中信号和槽取代了这些凌乱的函数指针，使得我们编写这些通信程序更为简洁明了。

所有从 `QObject` 或其子类（例如 `QWidget`）派生的类都能够包含信号和槽。当对象改变其状态时，信号就由该对象发射 (emit) 出去，这就是对象所要做的全部事情，它不知道另一端是谁在接收这个信号。这就是真正的信息封装，它确保对象被当作一个真正的软件组件来使用。槽用于接收信号，但它们是普通的对象成员函数。一个槽并不知道是否有任何信号与自己相连接。而且，对象并不了解具体的通信机制。

你可以将很多信号与单个的槽进行连接，也可以将单个的信号与很多的槽进行连接，甚至于将一个信号与另外一个信号相连接也是可能的，这时无论第一个信号什么时候发射系统都将立刻发射第二个信号。总之，信号与槽构造了一个强大的部件编程机制。

说实话对于像我这样的新手来说看着就蛋疼，想学会它没办法，我们还是简化一下概念吧：

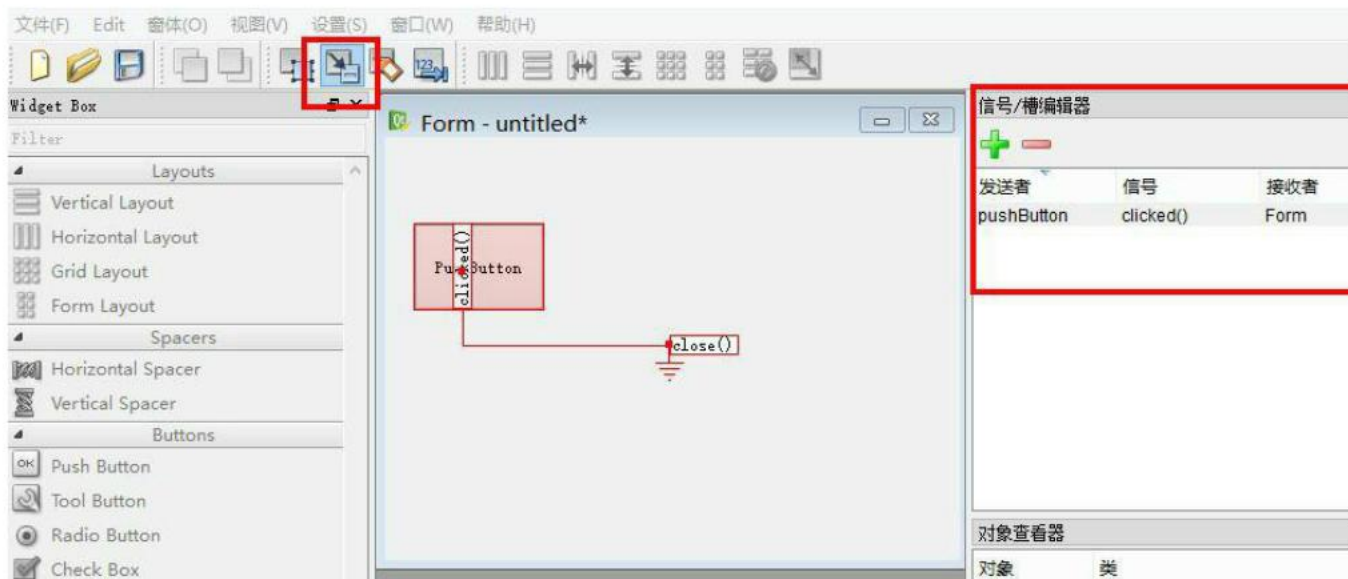
所有 `QObject` 类都可以使用信号槽，换句话说继承自 `pyqt` 中的类基本上都可以使用信号槽机制。当然非 `QObject` 也是可以通过其他一些办法来使用信号槽的。

仅仅有了信号和槽是不行的，我们还需要了解：

信号 (Signal)、槽 (slot)、连接 (connect)、动作事件 (action)、发射 (emit)、发送者、接受者等等一些列的知识。

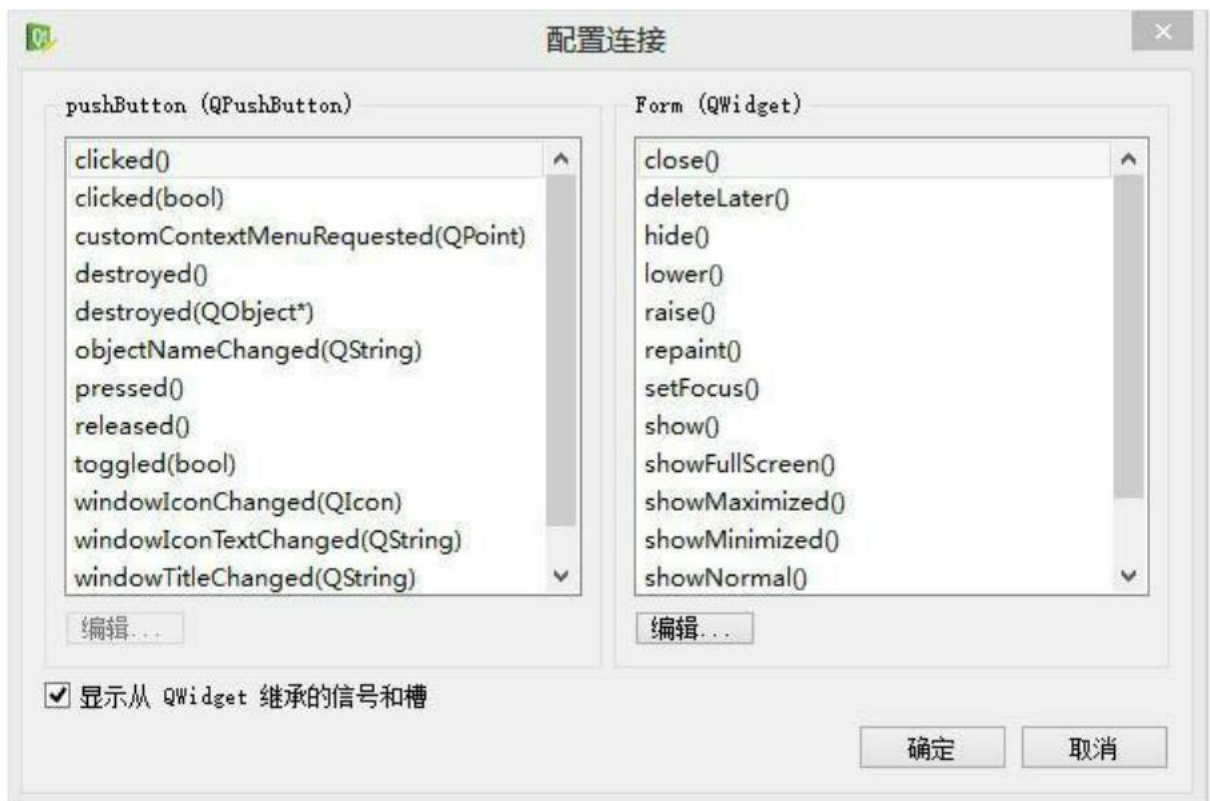
好吧，别搞的那么复杂行不行，我们还是学学该怎么用吧。

在 Qt Designer 中为我们提供了一些基本的信号槽方法，我们来看看：



点击工具栏上的“编辑信号/槽”，进入信号槽编辑模式，

我们可以直接在发送者（button）上按住不放拖动到接收者（Form 窗体）上既可以建立连接。



然后会弹出信号槽的配置连接。

左边是发送者（按钮）的信号（动作事件），右边是接收者（窗体）的槽（动作事件）

我们看一下编译后生成的代码：

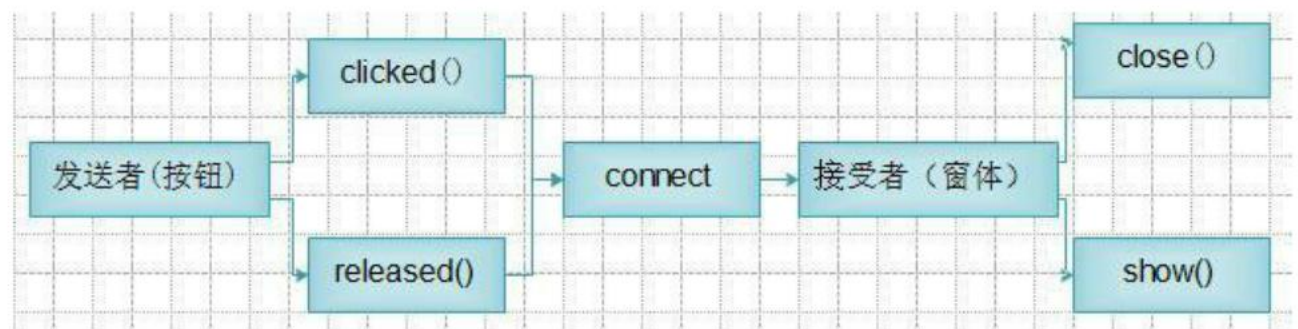
```
pushButton.clicked.connect(Form.close)
```

简单解释就是：当按钮点击之后关闭窗体。

实际却是：按钮.点击.链接(窗体.关闭)

其实应该是：按钮这个发送者，当按钮“点击”这个时间发生之后会发送一个信号出去，通过这段代码程序内部的通讯机制知道这个按钮的点击事情被连接到窗体的关闭事件上去了，然后通知接受者窗体，你该运行槽函数 **close** 了！

简单理解：比如我写了封信给你，送到邮局，邮局通过地址，转交给你。所以一个槽并不知道是否有任何信号与自己相连接。而且，对象并不了解具体的通信机制。



其实以上只是个人的浅显理解，具体的通信机制还是非常复杂的，想象一下邮局的工作就知道啦。

我们再来看看一些具体的代码：

```
self.pushButton.clicked.connect(Form.close)
```

按钮点击连接到窗口关闭

```
self.pushButton.clicked.connect(self.pushButton_2.hide)
```

按钮点击连接到按钮_2 隐藏

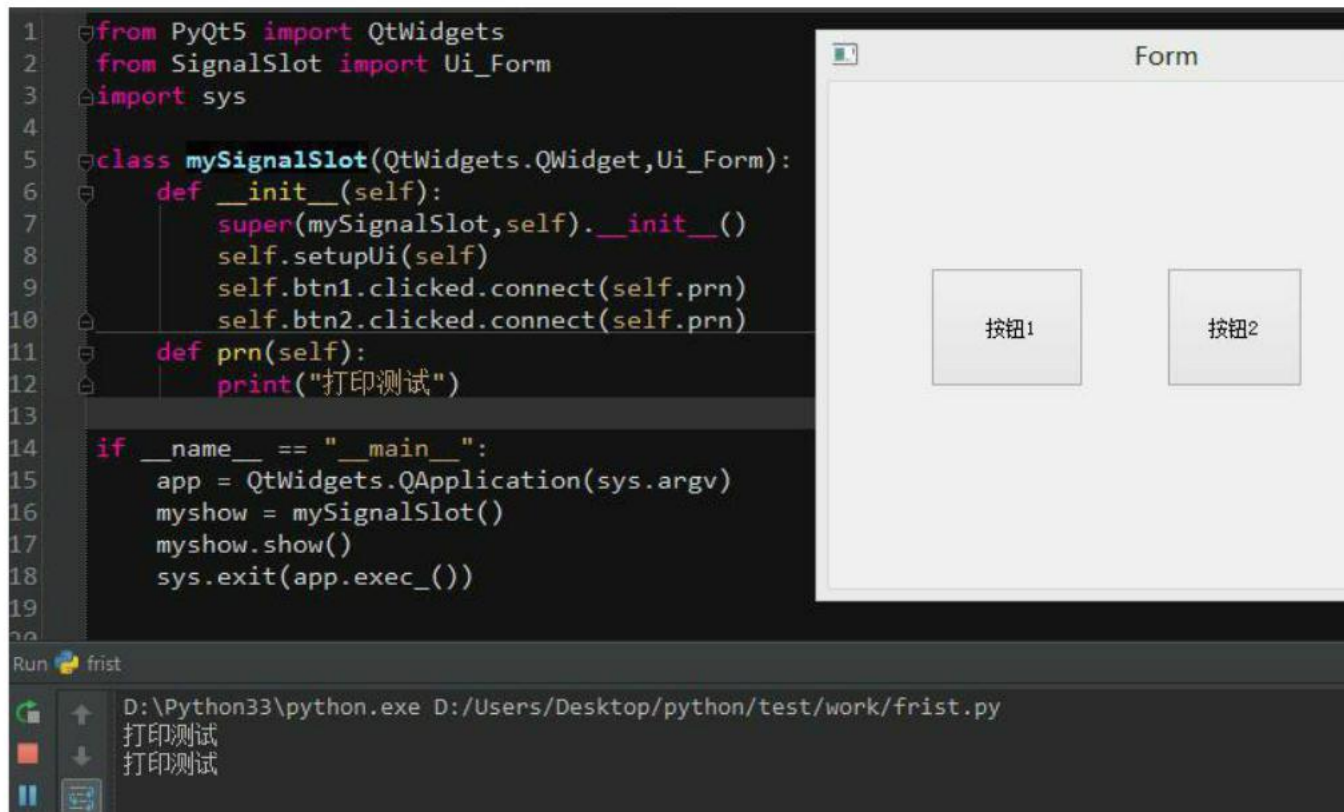
```
self.pushButton.destroyed.connect(self.pushButton_2.close)
```

按钮销毁连接到按钮_2 关闭

```
self.pushButton.released.connect(Form.update)
```

按钮重载连接到窗口更新

那么我们怎么能执行自己的“槽”呢？



通过之前学习的方法，我们继承了 `Ui_Form`，然后将按钮 1 和按钮 2 都连接到 `prn` 函数上，测试 OK。

对于初学者想要简单使用信号槽的机制，其实并不难，当然其中也有不少坑，还是多多测试为妙！

知识点：

在 PyQt 中接受者和发送者必须是个对象（实例）！

PyQt 中的控件中提供了很多信号和槽方法，大家可以多多使用 Qt Designer 设计参考！


槽其实就个函数（方法），Qt5 中的槽函数不在限定必须是 slot，可以是普通的函数、类的普通成员函数、lambda 函数等。编译期间就会检查信号与槽是否存在！

信号的 connect 连接最好放在__init__析构函数里面，这样只会声明一次连接，如果在类方法（函数中）使用的话，要记得 disconnect，否则 connect 会连接多次，导致程序异常。

信号槽函数不用加 ()，否则可能会导致连接异常。

PyQt5&python Gui 入门教程（10）Qt Designer 自定义信号 emit 及传参


```
1 from PyQt5 import QtWidgets,QtCore
2 from SignalSlot import Ui_Form
3 import sys,time
4
5 class mySignalSlot(QtWidgets.QWidget,Ui_Form):
6     _signal = QtCore.pyqtSignal() # 定义信号
7
8     def __init__(self):
9         super(mySignalSlot,self).__init__()
10        self.setupUi(self)
11        self.btn1.clicked.connect(self.prn) #按钮连接到prn槽函数
12        self._signal.connect(self.mysignalslot) #将信号连接到mysignalslot槽函数
13
14    def prn(self):
15        print("打印测试")
16        time.sleep(1)
17        print("延时1秒")
18        self._signal.emit() #发射信号
19
20    def mysignalslot(self): #自定义槽函数
21        print("我是槽")
22
23
24    if __name__ == "__main__":
25        app = QtWidgets.QApplication(sys.argv)
26        myshow = mySignalSlot()
27        myshow.show()
28        sys.exit(app.exec_())
29
```



上面的代码简单的自定义了一个信号，并连接到自定义的槽函数中。

其实看起来也不是很难的样子。

注意：声明信号必须在类属性中定义，也就是所有方法之前。

使用 emit 可以发射信号，然后通过 connect 将信号与槽函数连接。

下面我们来学习一下信号与槽之间的参数传递：


```
1 from PyQt5 import QtWidgets,QtCore
2 from SignalSlot import Ui_Form
3 import sys,time
4
5 class mySignalSlot(QtWidgets.QWidget,Ui_Form):
6     _signal = QtCore.pyqtSignal(str) # 定义信号,指定参数为str类型
7
8     def __init__(self):
9         super(mySignalSlot,self).__init__()
10        self.setupUi(self)
11        self.btn1.clicked.connect(self.prn) #按钮连接到prn槽函数
12        self._signal.connect(self.mysignalslot) #将信号连接到mysignalslot槽
13
14    def prn(self):
15        print("打印测试")
16        time.sleep(1)
17        print("延时1秒")
18        self._signal.emit("你是槽") #发射信号,传递参数
19
20    def mysignalslot(self,parameter): #自定义槽函数,接受参数
21        print(parameter)
22
23
24    if __name__ == "__main__":
25        app = QtWidgets.QApplication(sys.argv)
26        myshow = mySignalSlot()
27        myshow.show()
28        sys.exit(app.exec_())
29
30
31
32
33
34 # def mainwindows():
```

Run frist

D:\Python33\python.exe D:/Users/Desktop/python/test/work/frist.py

打印测试
延时1秒
你是槽

注意：当信号与槽函数的参数数量相同时，它们参数类型要完全一致。信号与槽不能有缺省参数。

当信号的参数与槽函数的参数数量不同时，只能是信号的参数数量多于槽函数的参数数量，且前面相同数量的参数类型应一致，信号中多余的参数会被忽略。此外，在不进行参数传递时，信号槽绑定时也是要求信号的参数数量大于等于槽函数的参数数量。这种情况一般是一个带参数的信号去绑定一个无参数的槽函数。


当然可以出传递的参数类型有很多种：str、int、list、object、float、tuple、dict 等等

至于信号槽使用注意事项可以百度一下

PyQt5&python Gui 入门教程（11）通用对话框

QMessageBox

```
1 from PyQt5 import QtWidgets,QtCore
2 from PyQt5.QtWidgets import QMessageBox
3 from SignalSlot import Ui_Form
4 import sys,time
5
6 class mySignalSlot(QtWidgets.QWidget,Ui_Form):
7     _signal = QtCore.pyqtSignal() # 定义信号,指定参数为str类型
8
9     def __init__(self):
10         super(mySignalSlot,self).__init__()
11         self.setupUi(self)
12         self.btn1.clicked.connect(self.msg)
13
14
15     def msg(self):
16         OK = QMessageBox.information(self,"这是标题"), ("""这是信息框"""),
17         QMessageBox.StandardButtons(QMessageBox.Yes |QMessageBox.No))
18
19 if __name__ == "__main__":
20     app = QtWidgets.QApplication(sys.argv)
21     myshow = mySignalSlot()
22     myshow.show()
23     sys.exit(app.exec_())
24
25
26
27
28
29
30
31
32
33
```



PyQt5 中为我们提供了很多默认信息框 QMessageBox，注意为方便使用需要导入模块。

QMessageBox 对话框包含类型只是图标不同其他无太大差别:

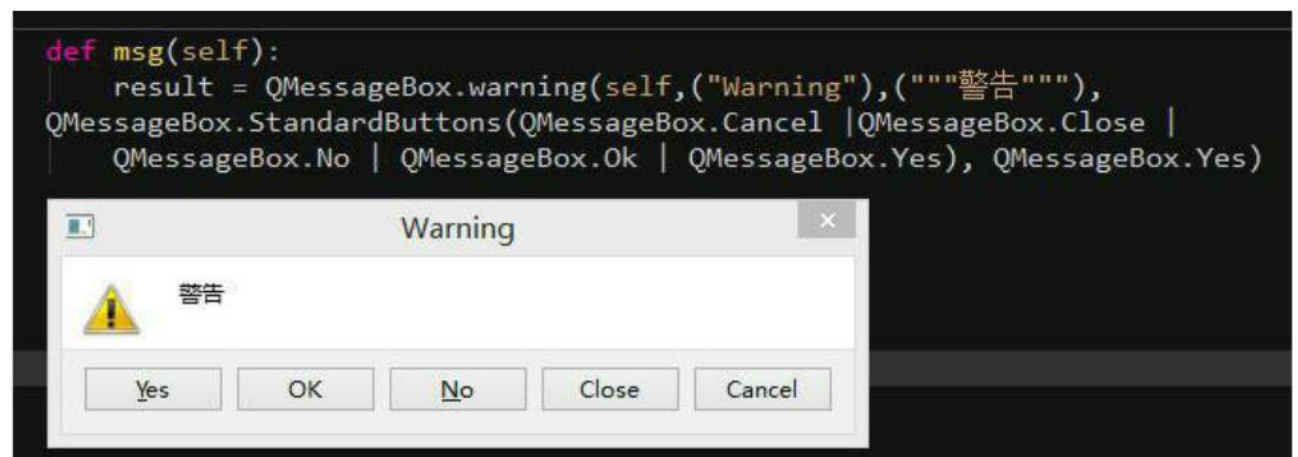
QMessageBox.information 信息框

QMessageBox.question 问答框

QMessageBox.warning 警告

QMessageBox.critical 危险

QMessageBox.about 关于

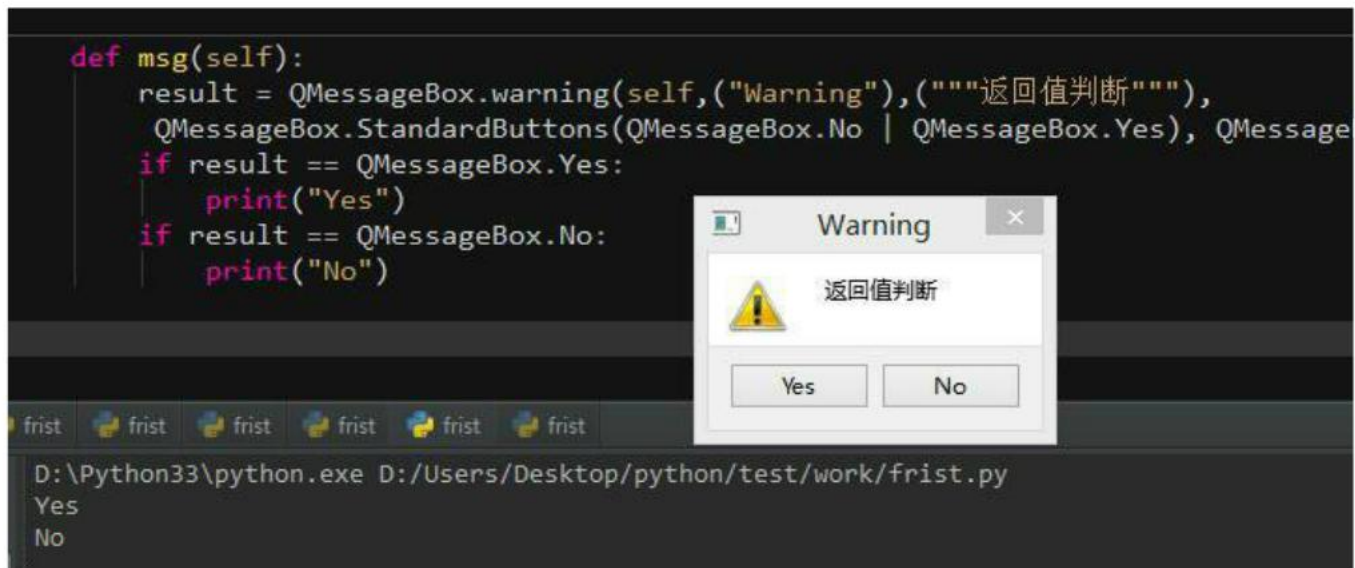


基本参数: (指定父组件, “标题”, “信息内容”, 要显示的按钮, 默认按钮)

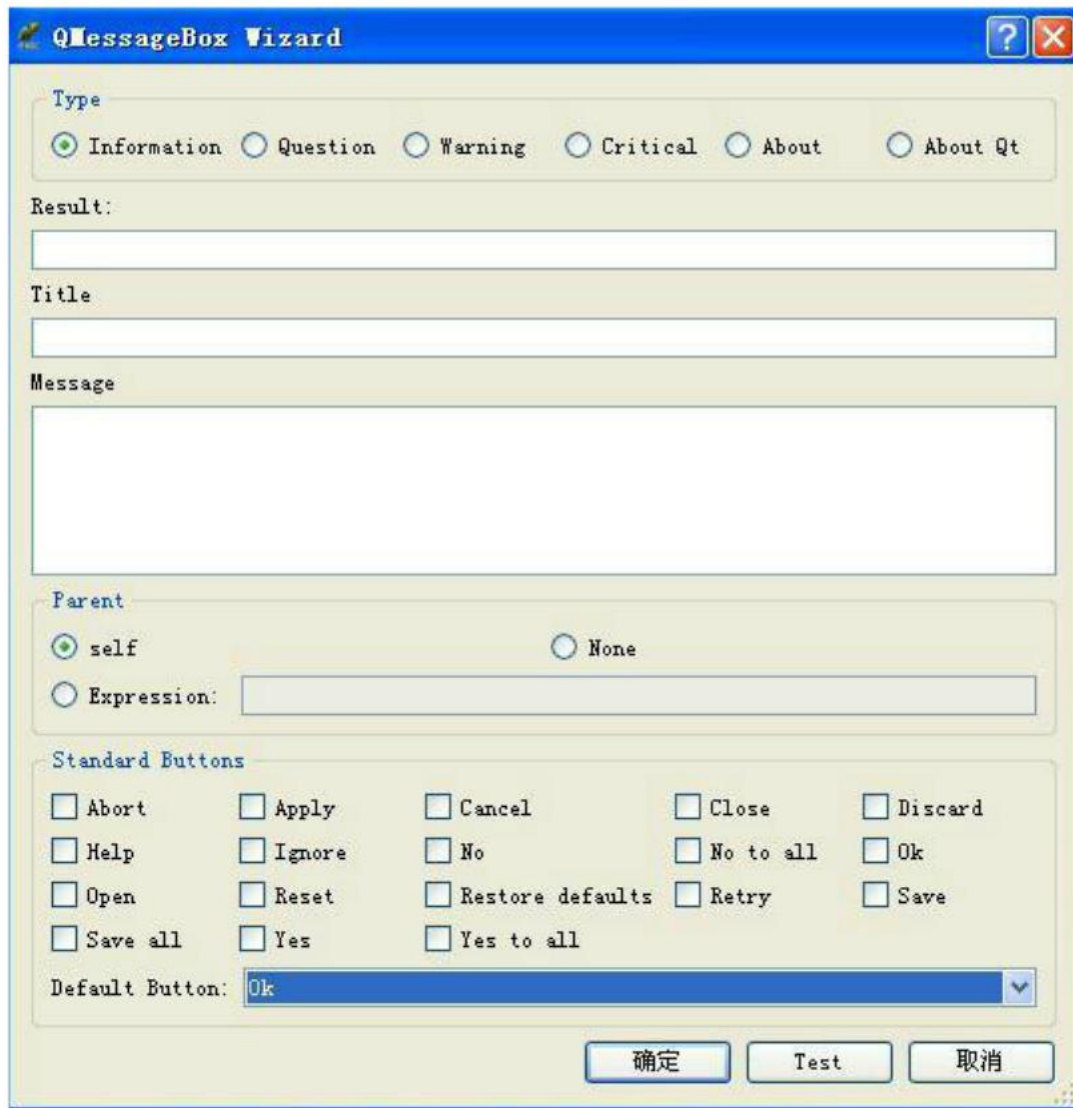
StandardButtons 默认要显示的按钮, 多个按钮使用 | 隔开。其中包含, 可以任意组合:

StandardButtons(QMessageBox.Abort | QMessageBox.Apply |
QMessageBox.Cancel |

QMessageBox.Close | QMessageBox.Discard | QMessageBox.Help |
QMessageBox.Ignore | QMessageBox.No | QMessageBox.NoToAll |
QMessageBox.Ok | QMessageBox.Open | QMessageBox.Reset |
QMessageBox.RestoreDefaults | QMessageBox.Retry
| QMessageBox.Save |
QMessageBox.SaveAll | QMessageBox.Yes | QMessageBox.YesToAll)

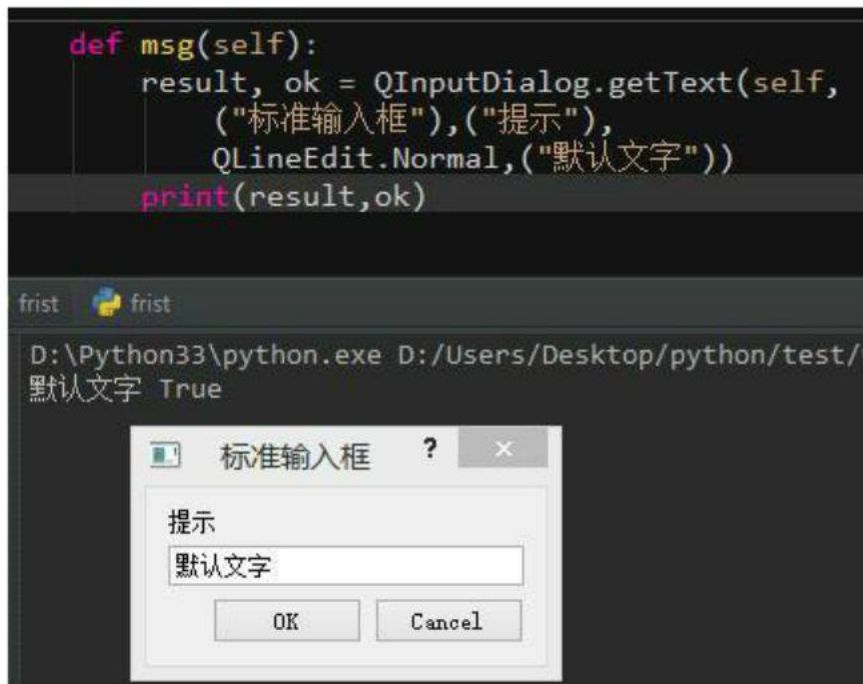


`result` 为返回值 默认为数值 我们可以通过上面的方法来判断用户点击了那个按钮！



之后来一张向导图，来自 eric。

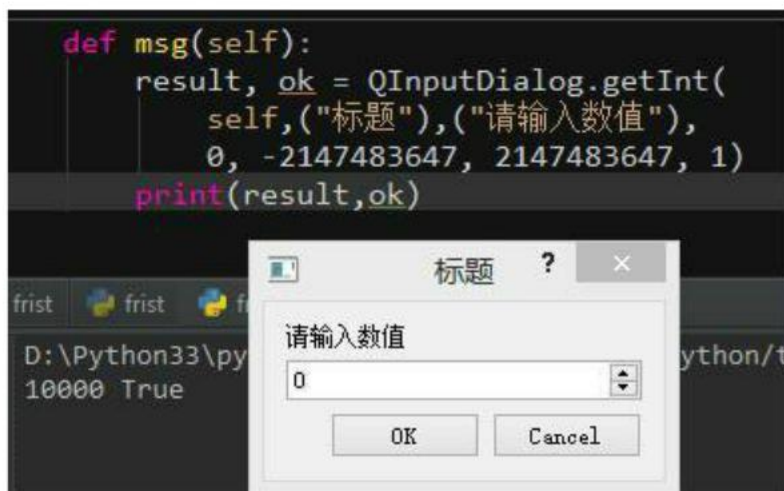
PyQt5&python Gui 入门教程（12）标准输入框 QInputDialog



标准输入文字框 `QInputDialog.getText()`

需要 `QInputDialog` 和 `QLineEdit` 模块,

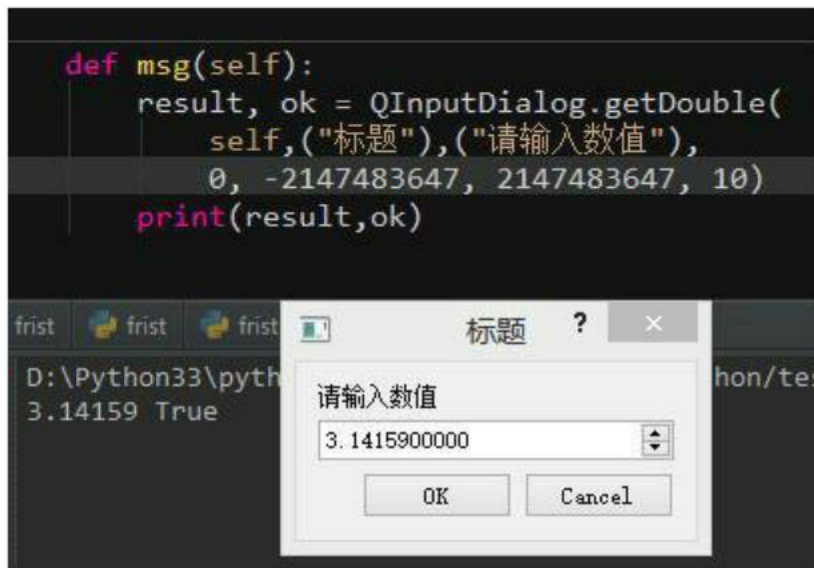
默认返回输入框文字和按钮 `bool` 值



标准整数输入框 `QInputDialog.getInt()`

默认返回输入框文字和按钮 `bool` 值

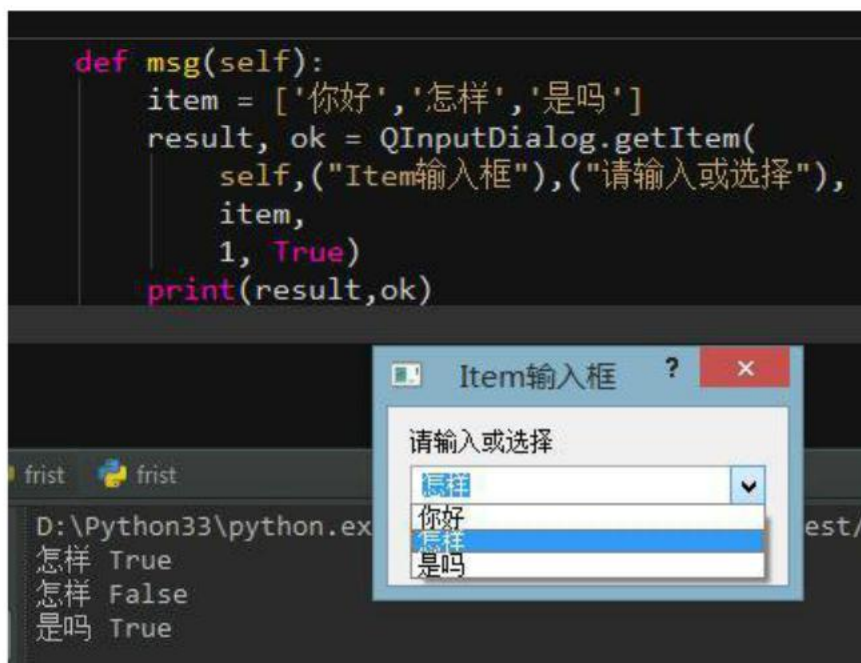
其中 0, -2147483647, 2147483647, 1 为默认数值, 数值范围, 和步长 (即按上下按钮时数值增加或减少多少)



标准浮点数输入框 `QInputDialog.getDouble()`

默认返回输入框文字和按钮 `bool` 值

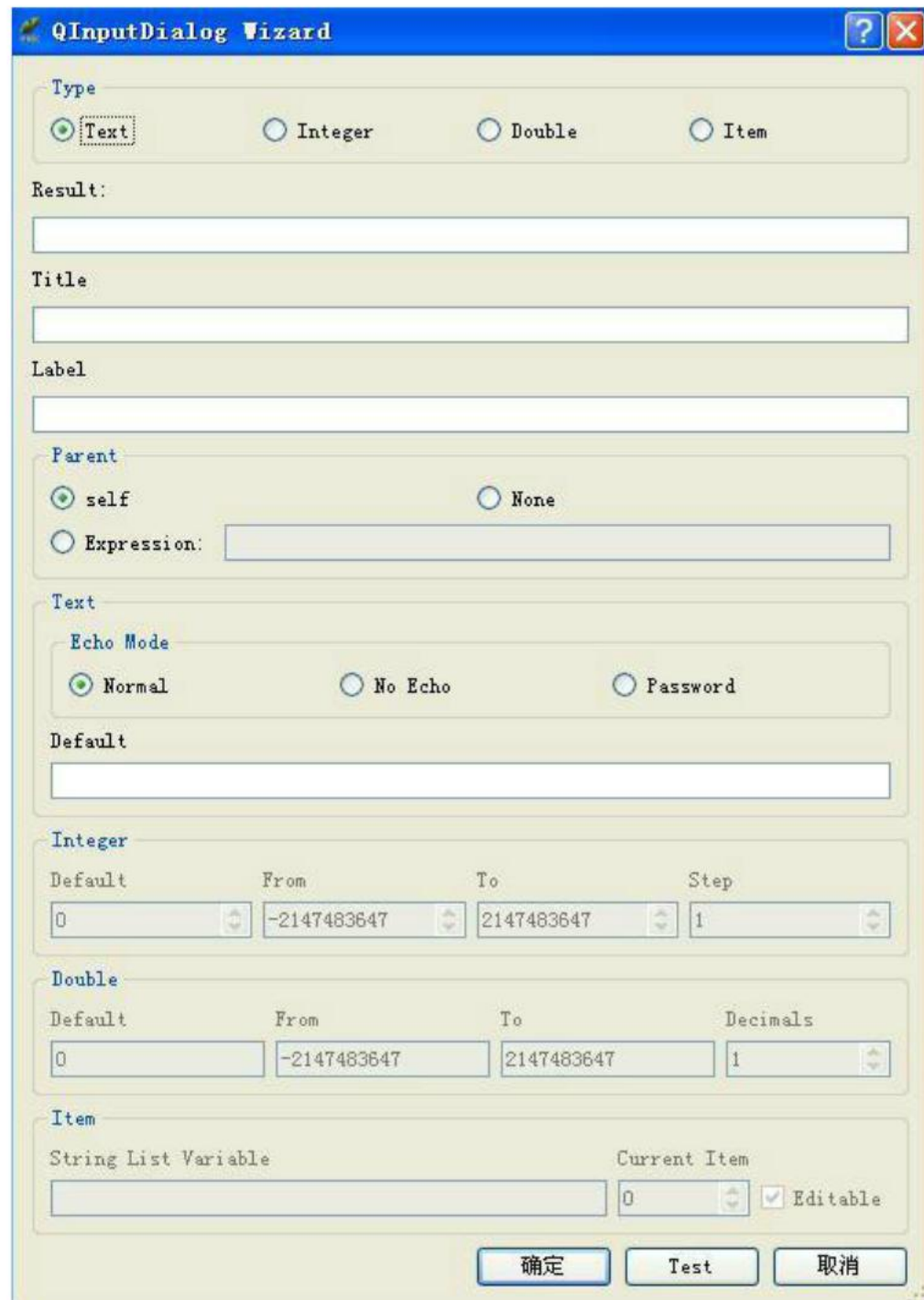
其中 0, -2147483647, 2147483647, 10 为默认数值, 数值范围, 最后 10 代表小数位数。



列表输入选择框 `QInputDialog.getItem()`

默认返回输入框文字和按钮 `bool` 值

条目添加为列表类型，其中 item 为自定义列表，1 为默认选中项目，True/False 列表框是否可编辑。

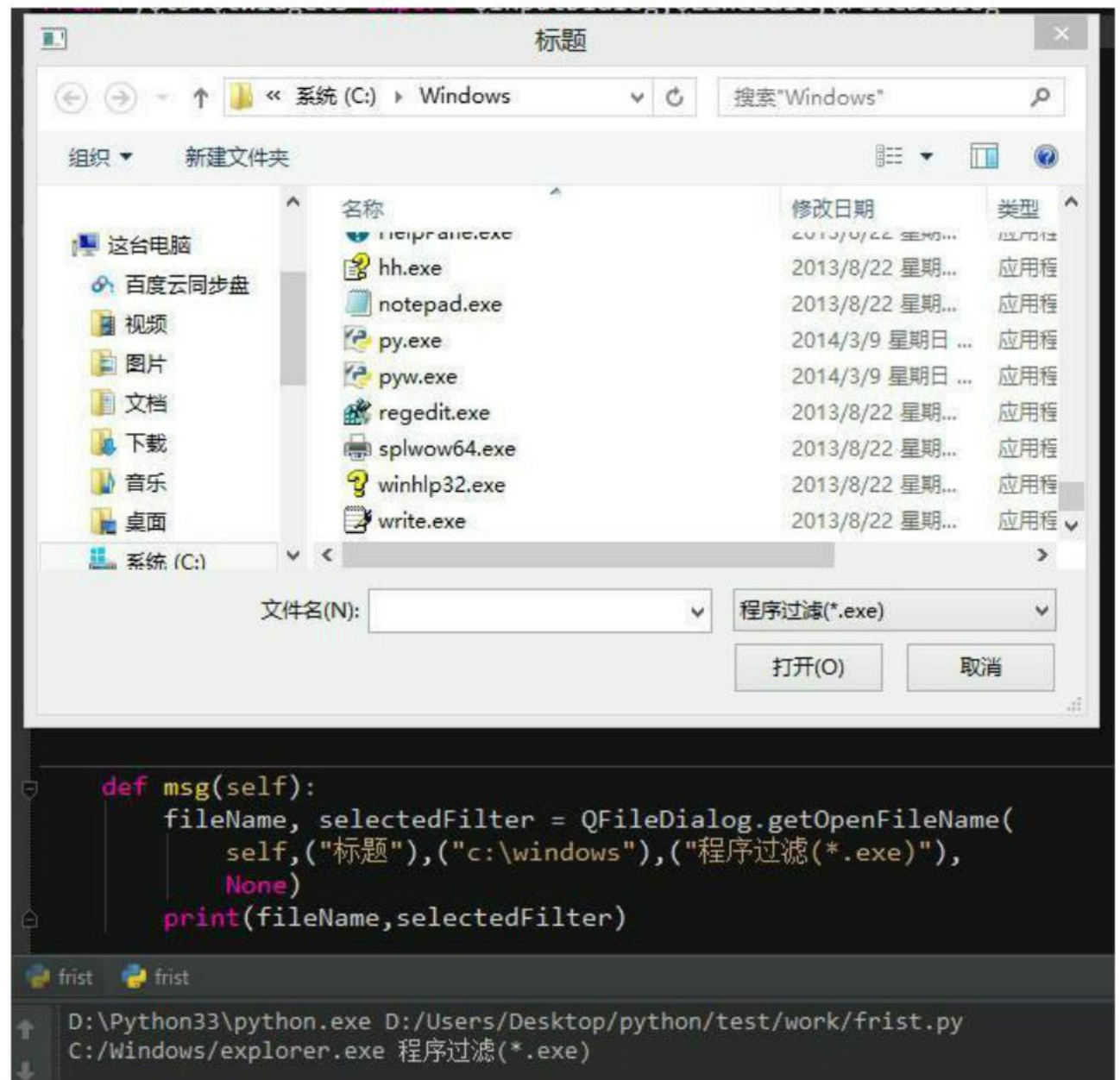


The image shows a Qt-style 'QInputDialog Wizard' dialog box. It has a blue title bar with a question mark icon and a close button. The dialog is divided into several sections for configuring an input dialog:

- Type:** Radio buttons for 'Text' (selected), 'Integer', 'Double', and 'Item'.
- Result:** A text input field.
- Title:** A text input field.
- Label:** A text input field.
- Parent:** Radio buttons for 'self' (selected) and 'None'. Below is an 'Expression:' text input field.
- Text:** A section for 'Echo Mode' with radio buttons for 'Normal' (selected), 'No Echo', and 'Password'.
- Default:** A text input field.
- Integer:** Fields for 'Default' (0), 'From' (-2147483647), 'To' (2147483647), and 'Step' (1).
- Double:** Fields for 'Default' (0), 'From' (-2147483647), 'To' (2147483647), and 'Decimals' (1).
- Item:** Fields for 'String List Variable' (empty), 'Current Item' (0), and a checked 'Editable' checkbox.

At the bottom are three buttons: '确定' (OK), 'Test', and '取消' (Cancel).

PyQt5&python Gui 入门教程（13）标准文件打开保存框 QFileDialog



单个文件打开 `QFileDialog.getOpenFileName()`

多个文件打开 `QFileDialog.getOpenFileNames()`

参数（指定父组件，“标题”，“默认打开路径”，“后缀名过滤器”）还有很多

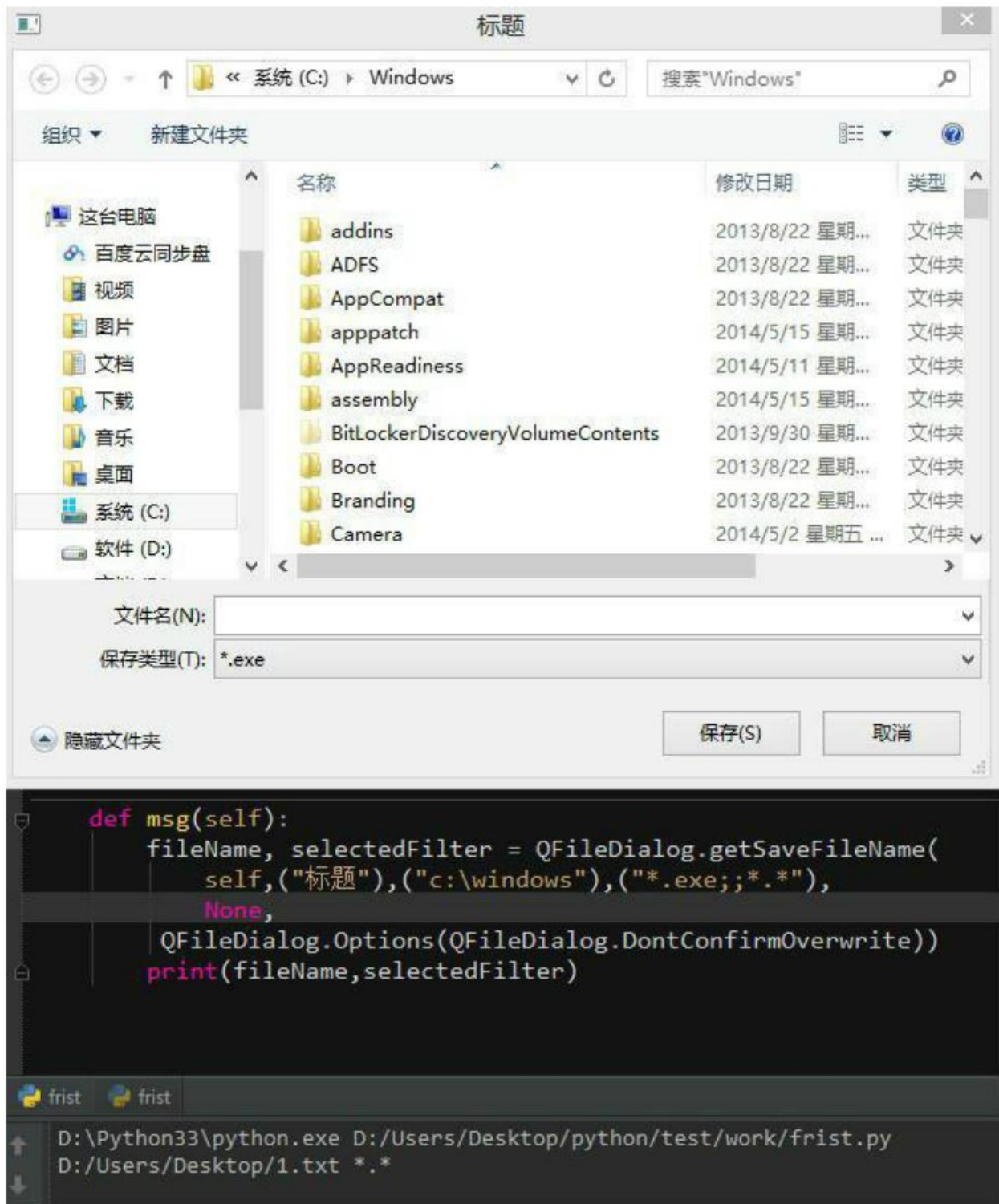
返回值 默认返回文件路径和当前过滤名

如果需要多种类型过滤，可以这样“jpg (*.jpg) ;; exe (*.exe) ;; (*.*)”用两个;;分割开。

getOpenFileName() 函数在 Windows 和 MacOS X 平台上提供的是本地的对话框，

而 QFileDialog 提供的始终是 Qt 自己绘制的对话框，而不都是调用系统资源 API。

注意多个文件打开，若要包含子目录还需要使用别的手段，自行百度吧！

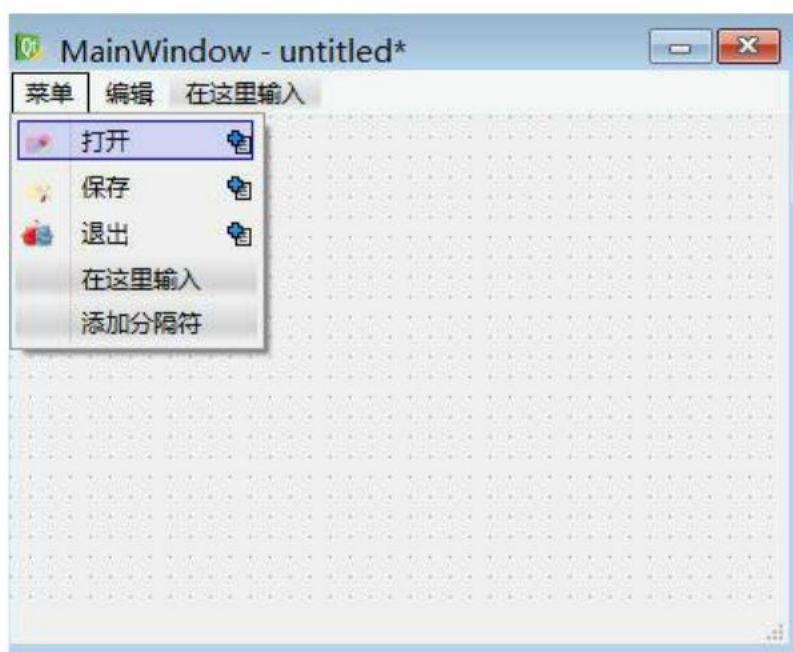


文件保存对话框 `QFileDialog.getSaveFileName()`

`QFileDialog.DontConfirmOverwrite` 覆盖提示?

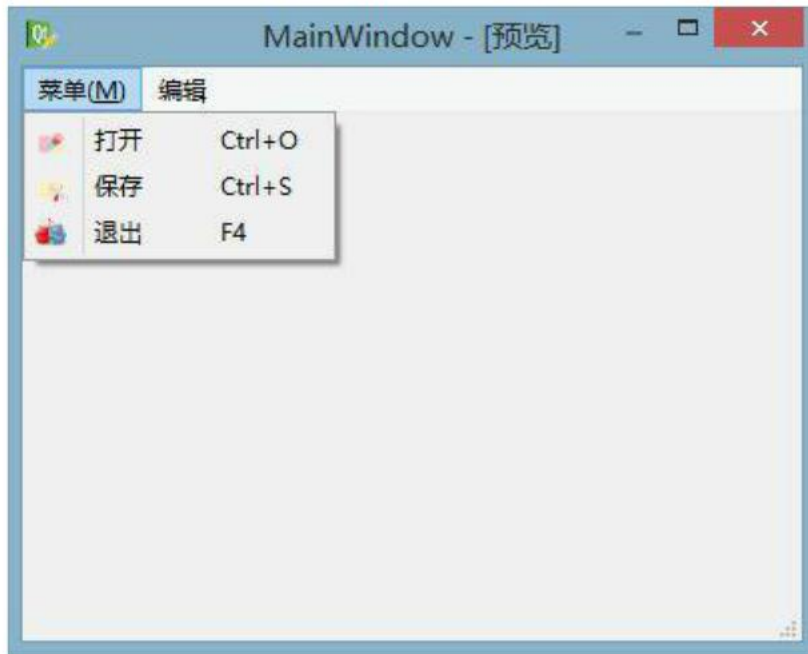
返回值同上, 同时具体生成文件内容还需要自己实现。

PyQt5&python Gui 入门教程（14） Qt Designer 主窗口 MainWindows



MainWindows 即主窗口，主要包含了菜单栏、工具栏、任务栏等

双击菜单栏上的“在这里输入”，然后录入文字，回车即可。注意要用回车键确认。



预览效果，并且加入了快捷键显示

我们可以这样做 菜单(&M) 编辑(&E) 来加入菜单的快捷按钮在预览中按 alt+m alt+e 可以看到效果

对象查看器

对象	类
MainWindow	QMainWindow
centralwidget	QWidget
menubar	QMenuBar
menu	QMenu
action_open	QAction
action_save	QAction
action_exit	QAction
menu_2	QMenu
statusbar	QStatusBar

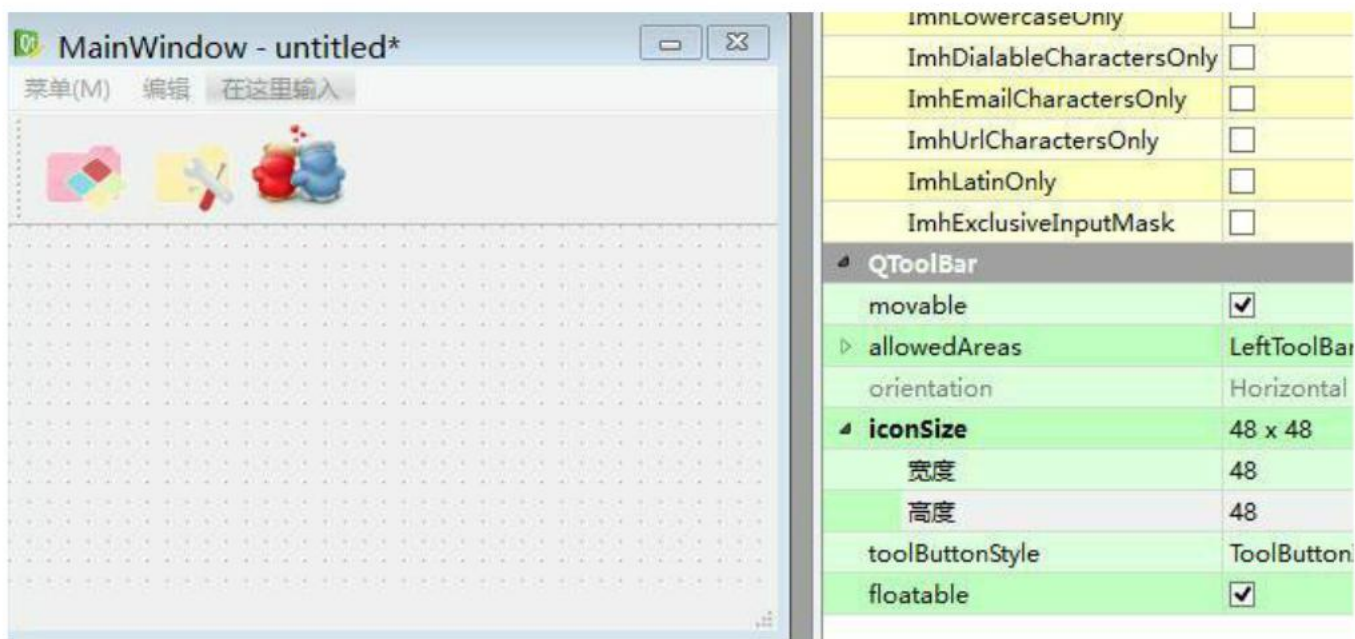
动作编辑器

名称	使用	文本	快捷键	可选的	工具提示
action_open	<input checked="" type="checkbox"/>	打开	Ctrl+O	<input type="checkbox"/>	打开
action_save	<input checked="" type="checkbox"/>	保存	Ctrl+S	<input type="checkbox"/>	保存
action_exit	<input checked="" type="checkbox"/>	退出	F4	<input type="checkbox"/>	退出

下拉菜单项 是 action 我们可以通过动作编辑器找到他们

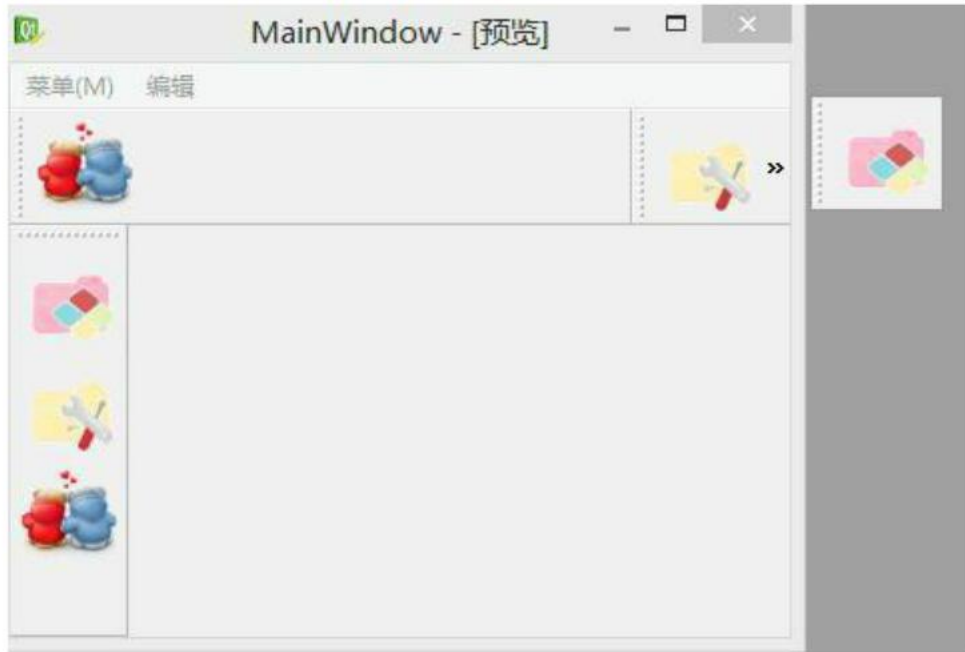


双击需要编辑的 action，可以对其进行设置并添加图标、快捷键等等。

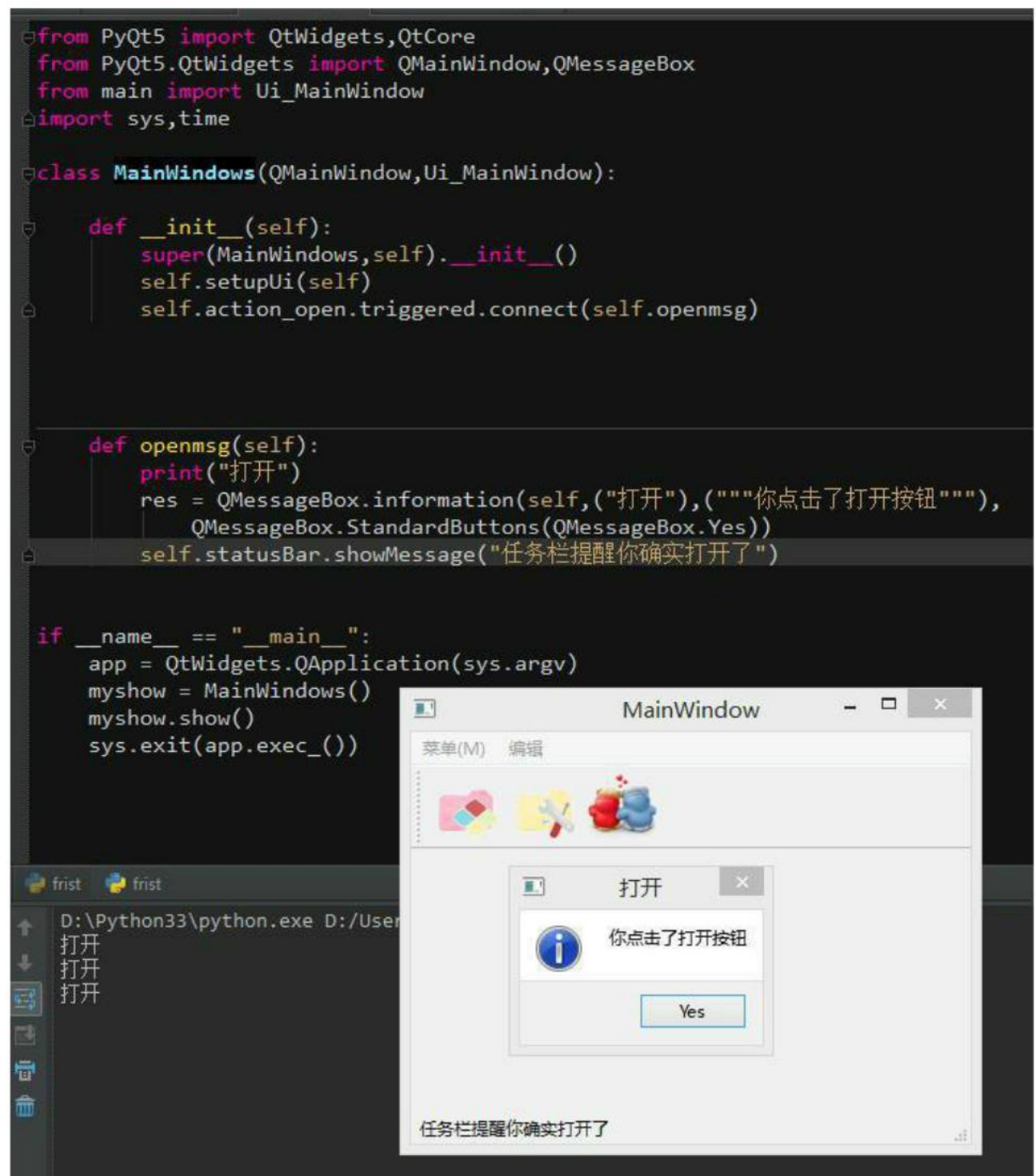


工具栏，默认在 PyQt5 中不显示工具栏，我们可以点击右键来添加工具栏，通过属性编辑器我们可以修改图标大小。

工具栏上也是 action，在动作编辑器上选择需要的 action 拖入即可。



我们可以添加多个工具栏，在预览中我们可以拖放工具栏，甚至是浮动出去，有点像 PS 的工具栏有木有？



我们来个综合实例看看

首先需要导入 QMainWindow 类 并继承 QMainWindow 生成自己的类。前面已经介绍很多不再复述！

然后将打开动作链接到自定义函数槽中,注意 action 的点击动作为 triggered。

槽函数中添加了一个通用对话框的信息框提示,

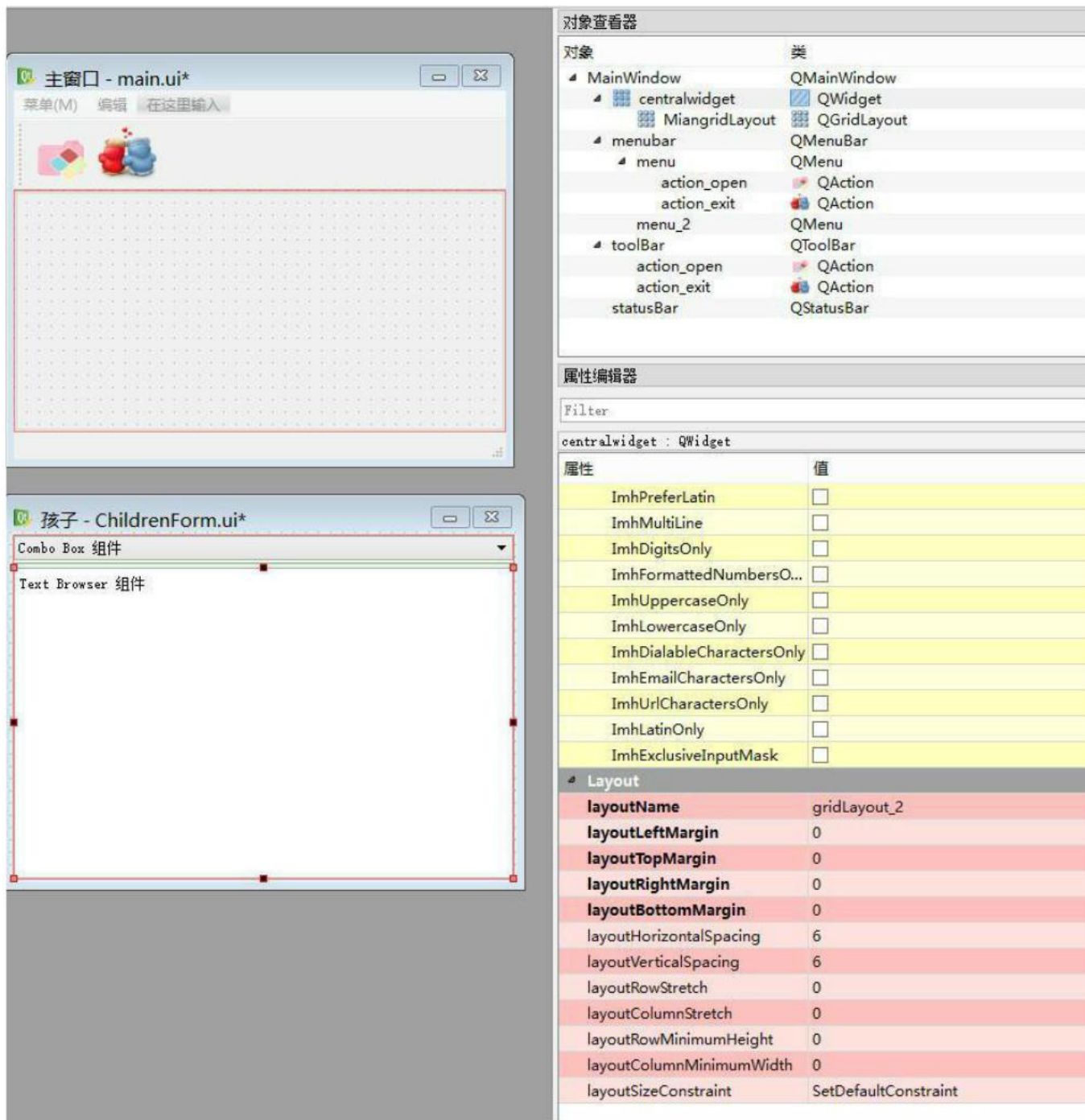
并且设置了任务栏提醒 `statusBar.showMessage()` 方法。

通过一个简单的例子可以看到 PyQt5&python Gui 开发结合 Qt Designer 非常的便捷和高效，其实入门也没那么难！

我们可以通过 Qt Designer 编译出来的文件去分析学习 Gui 构建，然后通过代码分离来实现自己的方法。

PyQt5&python Gui 入门教程（15）Qt Designer 主窗口动态加载 Widget

本人的教程比较基础，所用方法不够亮骚，仅供新手入门，若大家有更好的方法不胜赐教！



我们通过 Qt Designer 设计两个窗口，这里暂时命名为主窗口(mian)和孩子窗口(ChildrenForm)吧

我们在主窗口的空白中央添加一个栅格布局并命名为 MiangridLayout，等会需要将 ChildrenForm 放进去。

之前我们知道 mainwindows 默认有个 centralwidget 主层次布局，这里我们把 centralwidget 的边距需要设置一下见属性编辑器。

ChildrenForm 也用了栅格布局同时添加了一个横向布局和两个组件。

```

1  from PyQt5 import QtWidgets,QtCore
2  from PyQt5.QtWidgets import QMainWindow,QMessageBox
3  from main import Ui_MainWindow #导入主窗口类
4  from ChildrenForm import Ui_ChildrenForm #导入子窗口类
5  import sys
6
7  class children(QtWidgets.QWidget,Ui_ChildrenForm):
8      #继承QWidget和Ui_ChildrenForm, 我们设计的子窗口
9      def __init__(self):
10         super(children,self).__init__()
11         self.setupUi(self) #加载子窗口内容
12
13  class MainWindows(QMainWindow,Ui_MainWindow):
14      #继承QMainWindow 和 Ui_MainWindow 我们设计的主窗口
15      def __init__(self):
16         super(MainWindows,self).__init__()
17         self.setupUi(self) #加载主窗口内容
18         self.child = children()
19         self.MiangridLayout.addWidget(self.child)
20
21
22
23  if __name__ == "__main__":
24     app = QtWidgets.QApplication(sys.argv)
25     myshow = MainWindows()
26     myshow.show()
27     sys.exit(app.exec_())

```

首先导入必要的模块，并且导入我们自己设计的窗口类文件。

然后通过之前学过的方法来继承窗口类。

接着我们在主窗口后添加了两行代码

`self.child = children()` 生成子窗口实例 `self.child`

`self.MaingridLayout.addWidget(self.child)` 将实例加入到 Layout 用 `addWidget` 方法



最终显示效果，界面与逻辑完美分离，且实现了动态加载我们自己设计的子窗体。

```
class MainWindow(QMainWindow,Ui_MainWindow):

    def __init__(self):
        super(MainWindows,self).__init__()
        self.setupUi(self)
        self.child = children()
        self.action_open.triggered.connect(self.childshow)
        self.action_exit.triggered.connect(self.childhide)

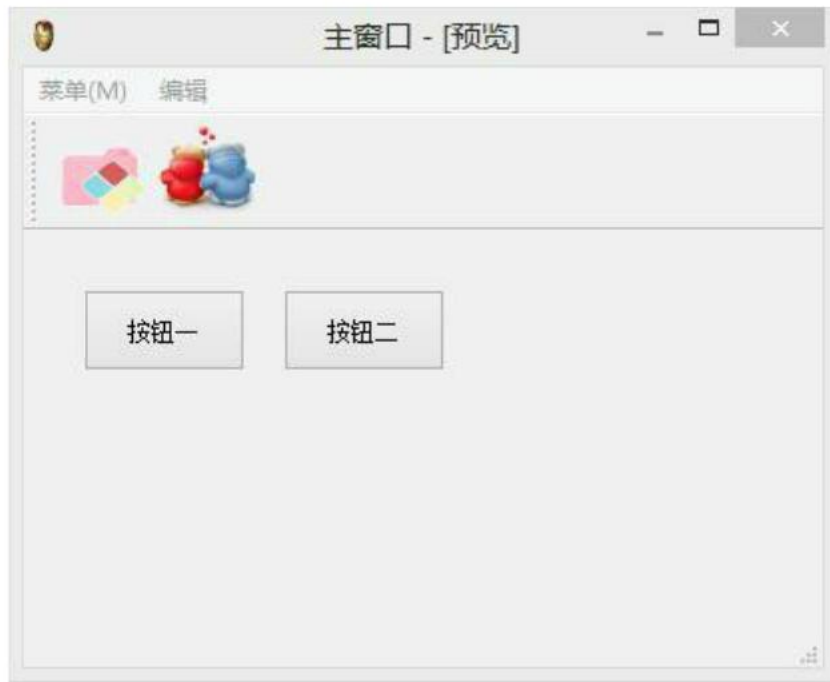
    def childshow(self):
        self.MiangridLayout.addWidget(self.child)
        self.child.show()

    def childhide(self):
        self.child.hide()
```

我们在改改代码，分别定义了两个函数，用来显示和隐藏子窗口，并将菜单按钮动作分别连接上去。

PyQt5&python Gui 入门教程（16）Qt Designer 主窗口之任务栏

上一篇我们提到了主窗口的任务栏功能，默认的任务栏只提供了一条信息框，但是很多时候我们需要分栏来显示不同的消息，这时候我们就需要自己去搞定了。



首先我们设计一个主窗体（为了方便沿用上一篇的设计），并添加任务栏。

```

from PyQt5.QtWidgets import QMainWindow
from main import Ui_MainWindow #导入主窗口类
import sys

class MainWindows(QMainWindow,Ui_MainWindow):

    def __init__(self):
        super(MainWindows,self).__init__()
        self.setupUi(self)
        self.statusBar.setStyleSheet("QStatusBar::item{border: 0px}")#去掉任务栏分割线

        self.label_1= QtWidgets.QLabel()#生成label控件
        self.label_1.setMinimumSize(QtCore.QSize(200,20))#设置label控件大小
        self.statusBar.addPermanentWidget(self.label_1,1)#将label控件加入到任务栏
        self.label_1.setText("我是label_1")#设置label文件

        self.label_2= QtWidgets.QLabel() #同上
        self.label_2.setMinimumSize(QtCore.QSize(100,20))
        self.statusBar.addPermanentWidget(self.label_2,1)
        self.label_2.setText("我是label_2")

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myshow = MainWindows()
    myshow.show()
    sys.exit(app.exec_())

```

下面我们来看看实现的步骤吧，

首先为了美观去掉任务栏分割线，然后生成 label 并设置大小，

然后用 addPermanentWidget 方法加入到任务栏，为了防止任务栏信息遮挡 label 文字

然后设置 label 要显示的字体来测试效果。



运行效果，so easy!

我们可以知道通过之前学习的方法我们继承窗体类，也学习了动态加入窗体，现在也知道还可以动态加入控件。

```

1  from PyQt5 import QtWidgets,QtCore
2  from PyQt5.QtWidgets import QMainWindow
3  from main import Ui_MainWindow #导入主窗口类
4  import sys
5
6
7  class MainWindows(QMainWindow,Ui_MainWindow):
8
9      def __init__(self):
10         super(MainWindows,self).__init__()
11         self.setupUi(self)
12         self.statusBar.setStyleSheet("QStatusBar::item{border: 0px}")#去掉任务栏的分割线
13
14         self.label_1= QtWidgets.QLabel()#生成label控件
15         self.label_1.setMinimumSize(QtCore.QSize(200,20))#设置label控件大小
16         self.statusBar.addPermanentWidget(self.label_1,1)#将label控件加入到任务栏
17
18         self.label_2= QtWidgets.QLabel() #同上
19         self.label_2.setMinimumSize(QtCore.QSize(100,20))
20         self.statusBar.addPermanentWidget(self.label_2,1)
21
22         self.action_open.triggered.connect(self.label1settext) #连接信号槽
23         self.action_exit.triggered.connect(self.label2settext) #连接信号槽
24         self.pushButton.clicked.connect(self.setalltext) #连接信号槽
25
26     def label1settext(self):
27         self.label_1.setText("我是label_1")
28
29     def label2settext(self):
30         self.label_2.setText("我是label_2")
31
32     def setalltext(self):
33         self.label_1.setText("同时显示吧")
34         self.label_2.setText("你确定? ")
35
36 if __name__ == "__main__":
37     app = QtWidgets.QApplication(sys.argv)
38     myshow = MainWindows()
39     myshow.show()
40     sys.exit(app.exec_())

```

结合之前学过的知识，我们在稍微改一改代码，增加 3 个函数，并连接到相应的按钮上，OK 我们来看看效果。



PyQt5&python Gui 入门教程（17）Qt Designer 按钮 PushButton

按钮 PushButton 是 Gui 开发中最常用的控件之一，通过按钮来进行交互并完成一些事情。

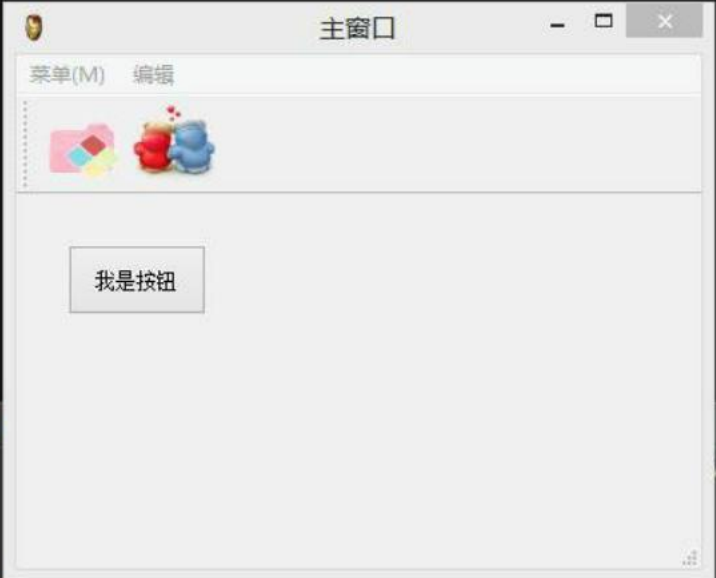
下面我们来看看 PushButton 的一些基本用法，


```
from PyQt5 import QtWidgets, QtCore
from PyQt5.QtWidgets import QMainWindow
from main import Ui_MainWindow #导入主窗口类
import sys

class MainWindows(QMainWindow, Ui_MainWindow):
    def __init__(self):
        super(MainWindows, self).__init__()
        self.setupUi(self)

        self.btn = QtWidgets.QPushButton(self.centralwidget) #创建按钮并加入到窗体中
        self.btn.setGeometry(QtCore.QRect(30, 30, 81, 41)) #设置相对坐标
        self.btn.setText("我是按钮") #设置按钮文字

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myshow = MainWindows()
    myshow.show()
    sys.exit(app.exec_())
```



简短的三行代码即可将一个按钮放置到窗体中，其中 `self.centralwidget` 是 `mainwindow` 自带的，代表中间空白区域。

然后通过 `setGeometry()` 设置按钮在窗体中的相对坐标，相对与窗体的坐标系而不是整个屏幕的坐标系。

最后设置一下文字 `setText()`，非常简单。

通过代码体验一下按钮的创建，当然通常我们会通过 `Qt Designer` 来设计，同时 `Qt Designer` 还提供了很多常用的设置项，

比如：字体 `font`、图标 `icon`、大小 `size`、快捷键 `shortcut`、提示信息 `toolTip` 等等，看看下图。



设置一个图标，代码：

```
icon1 = QtGui.QIcon()

icon1.addPixmap(QtGui.QPixmap("../ico/ironman.png"),
QtGui.QIcon.Normal, QtGui.QIcon.Off)

self.btn.setIcon(icon1)

self.btn.setFlat(True) 不显示边框，点击的时候还是会有边框提示
的

self.btn.setToolTip("设置提示内容")

self.btn.setEnabled(True) 是否可用

self.btn.setShortcut("F4") 设置快捷键
```

常用的事件有，一般用于连接槽函数。

clicked() 按钮点击

pressed() 按钮按下

released() 按钮释放

`toggled()` 切换按钮状态

判断按钮是否被按下可以用 `isDown()`

PyQt5&python Gui 入门教程（18）Qt Designer 按钮 PushButton 显示菜单



先上效果图，有的时候我们需要让按钮弹出一个自定义菜单，来让程序显得更酷更个性化，

下面我们就来看看代码吧。

```

7 class MainWindows(QMainWindow,Ui_MainWindow):
8     def __init__(self):
9         super(MainWindows,self).__init__()
10        self.setupUi(self)
11
12        self.menu = QtWidgets.QMenu()
13        self.m1=self.menu.addAction("新建")
14        self.m1.setIcon(QtGui.QIcon("../ico/batman.png"))
15        self.m1.triggered.connect(self.newfile)
16
17        self.m2=self.menu.addAction("删除")
18        self.m2.setIcon(QtGui.QIcon("../ico/spiderman.png"))
19        self.m2.triggered.connect(self.delfile)
20        self.pushButton.setMenu(self.menu)
21        self.pushButton.setFlat(True)
22        self.pushButton.setStyleSheet("QPushButton::menu-indicator{image:None}")
23
24
25    def newfile(self):
26        print("新建")
27
28    def delfile(self):
29        print("删除")
30
31
32    if __name__ == "__main__":
33        app = QtWidgets.QApplication(sys.argv)
34        myshow = MainWindows()
35        myshow.show()
36        sys.exit(app.exec_())

```

想要创建一个菜单 `self.menu = QtWidgets.QMenu()`,

然后通过后面的 `self.pushButton.setMenu(self.menu)` 将按钮重载。

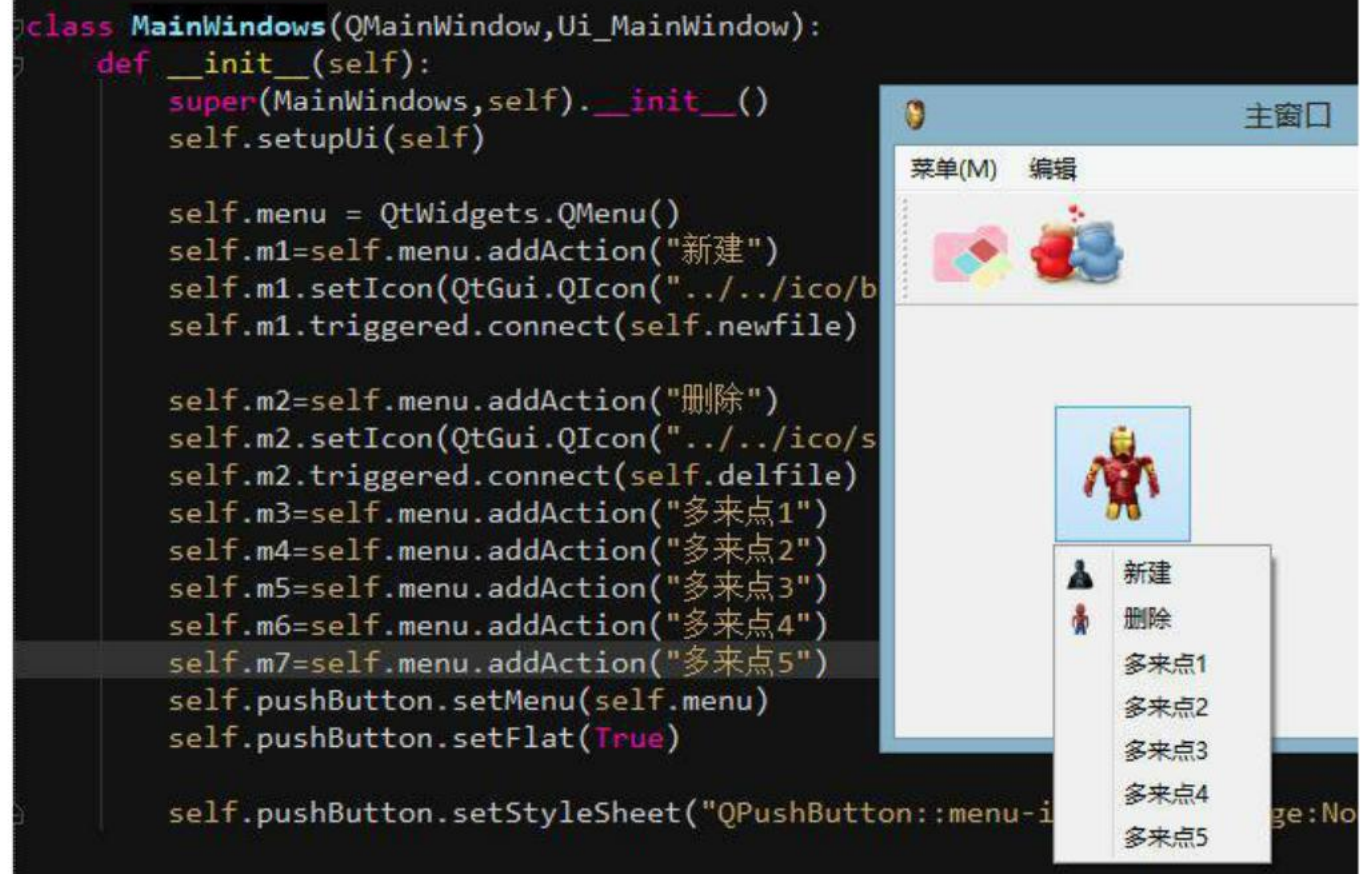
`self.m1, self.m2` 是两个 action,

通过 `addAction` 加入到 `self.menu` 中,

同时我们还为 action 设置了图标, 连接了信号槽。

`self.pushButton.setFlat(True)` 不显示按钮边缘。

`self.pushButton.setStyleSheet("QPushButton::menu-indicator{image:None}")` 去掉下拉提示箭头。



按钮菜单加长版！

之前讲过 mainwindows 中的菜单栏，大家可以研究一下 mainwindows 中的菜单代码，非常雷同。

PyQt5&python Gui 入门教程（19）Qt Designer 单选框 RadioButton

单选框 RadioButton 作为 Gui 开发常用组件之一，今天咱们来简单介绍一下，



如图所示，左边是在 Qt Designer 完成的，右边是效果预览。

第一排 `horizontalLayout` 布局中放入了三个单选框，第二排放入了两个，并将其中一个设置为 `clicked()`，

这样我们实现了两排单选框互不干扰，同时 `clicked()` 也实现了默认必选一项的功能。

对于单选框的判断也很简单，可以用 `isChecked()`。

```

from PyQt5 import QtWidgets,QtCore,QtGui
from PyQt5.QtWidgets import QMainWindow
from main import Ui_MainWindow #导入主窗口类
import sys

class MainWindows(QMainWindow,Ui_MainWindow):
    def __init__(self):
        super(MainWindows,self).__init__()
        self.setupUi(self)

        self.rbtn1.clicked.connect(self.radiobuttonclicked)
        self.rbtn2.clicked.connect(self.radiobuttonclicked)
        self.tbtn3.clicked.connect(self.radiobuttonclicked)

    def radiobuttonclicked(self):
        if self.rbtn1.isChecked():
            self.showmsg("赞")
        if self.rbtn2.isChecked():
            self.showmsg("支持")
        if self.tbtn3.isChecked():
            self.showmsg("顶")

    def showmsg(self,msg):
        self.statusBar.showMessage(msg)

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myshow = MainWindows()
    myshow.show()
    sys.exit(app.exec_())

```



我们看下实现代码，先将单选框动作连接到槽函数，然后自定义槽函数显示任务栏信息。

为了方便显示这里也单独定义了个 showmsg() 函数通过传参来显示任务栏信息。

如果要确定 2 个单选框被按下可以用 and 链接，列如：

```
if self.tbtn3.isChecked() and self.man.isChecked():
```

```
    self.showmsg("男人你顶了一下哦！")
```

前面我们说过定义类 class、函数 def 需要用:结尾，记得判断语句也需要用:结尾来标识，比如：if，while 等等。

PyQt5&python Gui 入门教程 (20) Qt Designer 复选框 CheckBox



复选框 CheckBox 相比单选框来说是可以重复多选的,这也是我们常用的组件之一。

它的用法和单选框基本相似,下面我们来看看简单的数相加的实现代码:

```

from PyQt5 import QtWidgets,QtCore,QtGui
from PyQt5.QtWidgets import QMainWindow
from main import Ui_MainWindow #导入主窗口类
import sys

class MainWindows(QMainWindow,Ui_MainWindow):
    def __init__(self):
        super(MainWindows,self).__init__()
        self.setupUi(self)

        self.cbox1.clicked.connect(self.cboxclicked)
        self.cbox2.clicked.connect(self.cboxclicked)
        self.cbox3.clicked.connect(self.cboxclicked)

    def cboxclicked(self):
        num = self.getNum()
        self.statusBar.showMessage(str(num))

    def getNum(self):
        n = 0
        if self.cbox1.isChecked():
            n += 100
        if self.cbox2.isChecked():
            n += 200
        if self.cbox3.isChecked():
            n += 300
        return n

if __name__ == "__main__":
    app = QtWidgets.QApplication(sys.argv)
    myshow = MainWindows()
    myshow.show()
    sys.exit(app.exec_())

```



同样将 checkbox 按钮动作连接到槽函数 checkclicked()

槽函数通过调用 getNum() 来获取当前按下的值。

也就是每点击一次 checkbox，getNum() 会计算一次当前那些按钮被按下，并将数值累加，然后返回数值。

通过 checkclicked() 槽函数的第二段代码将数字显示到 statusBar 任务栏上。当然需要通过 str 类型转换将数值转换成文本。